Spielend Programmieren lernen

Einführung in das kreative Programmieren mit dem Heimcomputer. Lernen Sie, Computerspiele selbst zu gestalten.

Von Karl-Heinz Koch. *Illustriert*

ratgeber

ht



Spielend Programmieren lernen



Im Computer-Bereich sind in unserer ht-Reihe bisher erschienen:

Computerrechnen – Schritt für Schritt	ht 146
Wie Transistoren funktionieren	ht 151
Datenverarbeitung – verständlich gemacht	ht 200
Taschenrechner	ht 292
Mikroprozessoren – Kleinstcomputer für alle	ht 338
Tischcomputer für Heim + Beruf	ht 415
Programmiersprache BASIC – Schritt für Schritt	ht 456
Schachcomputer – mein Hobby	ht 465
BASIC für Fortgeschrittene. 24 praktische Anwenderprogramme	ht 496
BASIC-Dialekte verstehen und vergleichen	ht 524
Lernen mit dem Homecomputer. BASIC-Programme für Schularbeiten	ht 525

Weitere Titel in Vorbereitung

Spielend Programmieren lernen

Von Karl-Heinz Koch

Humboldt-Taschenbuchverlag

humboldt-taschenbuch 526
Umschlaggestaltung: Christa Manner, München
Umschlagfoto: Peter Kasparides, München
Die Computerausdrucke im Innenteil stellte der Autor zur Verfügung.
Reinzeichnung der Abbildungen im Innenteil: Fritz E. Urich, München

Hinweis für den Leser:
Alle Angaben entsprechen dem Stand von 1985.
Sie sind von Autor und Verlag sorgfältig geprüft.
Dennoch kann eine Gewähr nicht übernommen werden.

Druck: Elsnerdruck, Berlin
Printed in Germany
ISBN 3-581-66526-3

1 2 3 * 87 86 85

Inhalt

1 Es ist leichter, als Sie denken!

Um es gleich vorweg zu sagen: Sie brauchen nichts Besonderes zu können oder zu wissen, kein Diplom als Elektroingenieur zu haben oder Magister der Mathematik zu sein. Programmieren werden Sie lernen, wenn Sie nur verstehen, daß ein Heimcomputer ein Elektrogerät wie jedes andere ist.

Der einzige, verwirrende, aber letztlich doch geringfügige Unterschied besteht darin, daß es nicht mehr für jede einzelne Funktion eine Taste gibt, sondern daß eine bestimmte Folge von Tasten zu drücken ist.

Um die vielen Knöpfe zu unterscheiden, sind sie mit Buchstaben bedruckt wie bei einer Schreibmaschine. Und die Tastenfolgen, die den Rechner arbeiten lassen, bilden, nacheinander gelesen, Wörter.

Alle möglichen Wörter zusammen bilden die Programmiersprache; oft wird auch Computersprache gesagt, als ob Computer sprechen und also auch denken könnten. Das ist natürlich blanker Aberglaube. Auch wenn bestimmte Kreise das dumme Wort von der »künstlichen Intelligenz« propagieren: Jede Maschine kann immer nur das nachmachen, was ihr Erbauer in sie hineingedacht hat.

Wie bei einer natürlichen Sprache müssen die Rechtschreibung und der Satzbau beachtet werden, wenn man eine längere Dienstanweisung an den Rechner verfassen will: ein Programm.

Und weil jeder, der sprechen kann, auch eine fremde Sprache lernen kann, deshalb kann auch jeder Programmieren lernen. Nun brauchen Sie aber nicht gleich Angst zu haben, Sie müßten endlos Vokabeln und Grammatik pauken. Computer sind so primitiv, daß man, wie bei einem gut abgerichteten Hund, mit einer handvoll Kommandowörtern auskommt.

Als ich vor vier Jahren die Zeit gekommen sah, mich mit diesen neuen kleinen Computern zu beschäftigen, über die immer mehr geredet wurde, meldete ich mich für einen Kurs an, um die Programmiersprache BASIC (sprich: beeßik) zu erlernen.

Schließlich saß ich zwischen Studenten, Mathematikern und Elektrotechnikern und war ungeheuer beeindruckt, wie kompliziert und wissenschaftlich das alles aussah. Da wurden *Struktogramme* entworfen und *Flußdiagramme* gezeichnet und die raffiniertesten mathematischen Tricks vorgeführt.

Am dritten Abend bekamen wir dann endlich ein wenig darüber zu hören, wie man BASIC ganz praktisch verwendet, und sollten nach einer Stunde Vorlesung unser erstes Programm schreiben.

Nach Eingabe von Länge, Durchmesser und Wandstärke eines Bleirohres sollte dieses Programm das Gewicht errechnen. Ich glaube, den Mathematikern hat das richtig Spaß gemacht. Ich habe mich gelangweilt. Allerdings wußte ich nach dieser lehrreichen Unterrichtsstunde, daß man die Grundbegriffe von BASIC an einem Abend erlernen kann.

Als wir in der folgenden Woche ausrechnen sollten, nach wievielen Minuten eine Tasse Kaffee mit einer Temperatur von 75 Grad Celsius auf die Zimmertemperatur von 21 Grad abgekühlt ist, wenn in jeder Minute 12% der Wärme abgegeben wird, entschied ich, daß diese Art, Programmieren zu lernen, für mich kalter Kaffee ist.

Ich kaufte mir einen Heimcomputer und spielte mit den BASIC-Wörtern herum. Das hat zu Anfang manche schlaflose Nacht gekostet, aber ich habe dabei mehr über das Programmieren gelernt, als mir die schlauesten Mathematikaufgaben hätten vermitteln können.

Deshalb ist dieses Programmierbuch keine Mathematikfibel. Und wenn Ihnen bunte Bilder, Bewegung und ein paar synthetische Töne auch mehr Spaß machen, als Formeln, Gleichungen und Theorien, dann werden Sie sicher nicht zu kurz kommen und »spielend Programmieren lernen«.

1.1 Stolpersteine

Sie sind wahrscheinlich schon stolzer Besitzer eines Heimcomputers; denn Programmieren kann man nicht als Trockenschwimmer

lernen. Sie müssen schon hineinspringen ins kühle Naß von Bits und Bytes, und Sie werden sich bald wie ein Fisch im Wasser fühlen.

In diesem Buch werden Sie nichts über Technik erfahren und über keinem Fachchinesisch rätseln müssen. Wenn Sie sich einen Videorecorder kaufen, dann lernen Sie auch nichts über Frequenzmodulation und Hifi-Technik, nur um ein Fernsehprogramm aufzuzeichnen.

Für mich ist der Heimcomputer ein kreatives *Freizeitspielzeug.* Jedenfalls habe ich bis heute noch keine wirklich zweckorientierte Anwendung eines Rechners im Haushalt kennengelernt. Erst wenn Sie Arbeiten von *geschäftlichem* Umfang zu erledigen haben, viel Korrespondenz, Archivierungen oder ähnliches, kann sich ein Computer nützlich machen. In der Regel aber ist er nicht mehr und nicht weniger als ein amüsanter Spielkamerad.

Handbücher für Heimcomputer (wie leider oft auch für andere Elektrogeräte) lesen sich meist wie die Rätselseite der Wochenendzeitung. Sie sind offensichtlich von Technikern verfaßt, die ihrer Freundin ins Ohr flüstern, »mein induktiver Blindwiderstand geht gegen Null«, wenn sie ihr eine Liebeserklärung machen.

Trotzdem müssen wir uns mit der auf das Praktische konzentrierten Welt der Techniker ein wenig einlassen, wenn wir uns an den Heimcomputer setzen. Und wir müssen es hinzunehmen lernen, daß der »Kolonialismus« heutzutage als technische Norm verbreitet wird. Für liebgewonnene nationale Eigenheiten bleibt kein Platz, wenn z. B. amerikanische Konstrukteure Ihre Maschine gebaut haben.

So müssen Sie bei den meisten Gerätetypen auf die Umlaute (ä, ö und ü) sowie auf das »ß« verzichten. Außerdem werden Sie, auch später noch, immer wieder »Z« und »Y« verwechseln, weil bei der üblichen (amerikanischen) Computertastatur diese beiden Tasten im Vergleich zur DIN-Tastatur vertauscht sind.

Daß die Dezimalstellen mit einem Punkt (.) statt mit einem Komma (,) markiert werden, daran haben uns schon die Taschenrechner gewöhnt. Mit den Heimcomputern müssen wir uns jedoch an eine Reihe weiterer ungewohnter Zeichen gewöhnen, z. B. das Betrag- oder At-Zeichen (@), auch Klammeraffe geschimpft, das Num-

mernzeichen (#) Raute, das Dollarzeichen ($) für »String« und ganz fremdartige Zeichen für die mathematischen Operationen. Von alten Freunden wie dem § hingegen müssen wir womöglich (?) für immer Abschied nehmen.

Wenn Sie noch auf einer Reiseschreibmaschine, Marke Typenhebel, nach der Adlermethode (Einkreisen und Zustoßen) Maschineschreiben gelernt haben, dann müssen Sie sich eventuell noch abgewöhnen, ein kleines L (l) statt einer Eins (1) und ein großes O (Oh) statt einer Null (0) zu tippen. Der Rechner ist in diesen Dingen ganz kleinkariert! Besonders Verwechslungen von Oh und Null sorgen beim Programmieren immer wieder für Kurzweil. Die Techniker streichen deshalb die Null zur Unterscheidung vom O (Oh) zusätzlich durch (weil sie aber auch gar keinen Wert hat).

Genauso genau nimmt es die Maschine mit allen anderen Zeichen natürlich auch. Das bereitet gerade dem Anfänger viel Verdruß. Kein einziges Zeichen darf fehlen, nicht ein klitzekleines zu viel dastehen. Besonders die so unscheinbar aussehenden Satzzeichen (,;.:), die selbst in Briefen gebildeter Leute manches Mal zu kurz kommen, übernehmen wichtige Aufgaben.

Achten Sie deshalb konzentriert auf jedes noch so unscheinbare Zeichen, wenn Sie ein Programm schreiben oder selbst entwerfen.

1.2 BASIC-Babel

Wenn Sie schon einmal in der Schweiz waren (oder dort zu Hause sind), kennen Sie den Wert der Behauptung, dort würde Deutsch gesprochen. Mit BASIC ist das nicht viel anders. Sicher, das Grundkonzept ist immer das gleiche; die wesentlichen Befehle, die umfassende Grammatik, lassen ein BASIC-Programm erkennen. Wenn Sie aber ein BASIC-Programm von Computermarke A mit einem von Marke B vergleichen, fühlen Sie sich leicht wie ein Berliner in Basel.

Konsequenterweise spricht man von verschiedenen *BASIC-Dialekten,* die im Kern, manche sprechen von »Hoch-BASIC«, gleich sind, aber im Detail deutliche Unterschiede zeigen; ähnliche Befehle haben andere Namen, werden in einer abweichenden Form geschrieben oder stehen gar nicht zur Verfügung.

Im mathematischen Kern sind alle BASIC-Dialekte gleich. Wenn es aber um Grafik, Ton oder die Kontrolle von Steuerknüppeln *(Joysticks)* geht, dann werden die Abweichungen sehr groß. Diese Funktionen werden von speziellen Bausteinen besorgt; und da verwenden unterschiedliche Computermodelle nicht nur verschiedene Prozessoren, sondern jeder Hersteller kocht auch sein eigenes BASIC-Süppchen.

Im wesentlichen werden in diesem Buch nur BASIC-Wörter verwendet, die praktisch bei allen Heimcomputern in gleicher Weise vorkommen. Da wir aber beim Programmieren unseren Spaß haben wollen und deshalb nicht auf Grafik verzichten können, trennen sich die Programmwege doch immer wieder.

Ein Konsortium *japanischer* Hersteller ist vor einiger Zeit mit einem Standard für Heimcomputer auf den Markt gekommen. Die Zauberformel lautet MSX. Die Norm bezieht sich nicht nur auf Stecker und Buchsen, so daß jedes MSX-Gerät mit jedem anderen MSX-Gerät zusammenpaßt, also ein MSX-Drucker von der Firma A, eine MSX-Diskettenstation von Firma B, eine Computerkonsole von Firma C und Programme von Firma D. Auch das BASIC ist vollkommen einheitlich und heißt natürlich MSX-BASIC.

Programmbeispiele in diesem Buch sind deshalb in *MSX-BASIC* geschrieben. Zusätzlich werden Programme in *Atari-BASIC* dazugestellt, weil die Grafikmöglichkeiten der Atari-Geräte bis heute im Bereich der Heimcomputer ihresgleichen suchen. Und schließlich wird *Commodore* berücksichtigt, eine Reminiszenz an die große Verbreitung dieser Maschine.

Da es wirklich nicht möglich ist, darüber hinaus weitere, womöglich alle anderen Dialekte zu berücksichtigen, finden Sie im Anhang eine Übersichtstabelle, in der die wichtigsten, in diesem Buch vorkommenden BASIC-Wörter in den Dialekten verschiedener Computermodelle nebeneinandergestellt sind. Mit dieser Tabelle und Ihrem Handbuch werden Sie die Programmbeispiele leicht für Ihren Rechner anpassen können. Eine gute Übung übrigens, denn BASIC lernen heißt auch, sich in BASIC-Dialekten zurechtzufinden.*

---

* Wir empfehlen in diesem Zusammenhang unseren Band »BASIC-Dialekte verstehen und vergleichen« (ht 524), einen Titel für Fortgeschrittene vom Humbold-Taschenbuchverlag, München.

2 BASIC-Basis

BASIC ist die Abkürzung für Beginners All Purpose Symbolic Instruction Code, was frei übersetzt »Allzwecksymbolsprache für Anfänger« heißt.

BASIC selbst ist eine Art Programm, oder genauer ein Übersetzer (Interpreter). Sie wissen sicher, daß der Rechner wie jedes andere Elektrogerät nur zwei Daseinsformen kennt: Strom (Ein) und Nichtstrom (Aus), wie ein Schalter. In mathematischen Zeichen werden diese Zustände des Schalters als 1 und 0 dargestellt; der Informationsgehalt eines Schalters, eine Ja/Nein- oder Aus/Ein- oder 0/1-Entscheidung, heißt ein *Bit.*

Sehr, sehr klein sind diese Schalter, und so ein Heimcomputer hat beachtlich viele davon – Modelle gängiger Größe mehr als eine halbe Million. Deshalb werden auch mehrere Bit gleichzeitig bearbeitet. Bei heute üblichen Heimcomputern sind es acht. Acht Bit entsprechen einem *Byte.* Personalcomputer bearbeiten schon sechzehn, immer mehr sogar 32 Bit gleichzeitig.

Der BASIC-Interpreter übersetzt also die Befehle, die wir eintippen, in diese endlose Schlange von Einsen und Nullen, die in Form von Spannung und Nichtspannung mit Elektronengeschwindigkeit durch Chips und Leitungen flitzen.

BASIC kann gut mit einer Kindersprache verglichen werden. Wenn ein Baby sagt: »Mama, Appa mam-mam«, brauchen wir nur zu wissen, daß mit »Appa« ein Apfel gemeint ist, um den ganzen Sinn des Kommandos zu verstehen. Das ist bei BASIC nicht anders.

Nehmen wir an, der Rechner soll folgende Anweisung bekommen: Wenn A den Wert 20 hat, dann laß B den Wert 5 annehmen.

Bringen wir diesen schönen Satz auf Technikerformat, so lautet er: Wenn A=20, dann B=5. Jetzt müssen wir nur noch eine atlantische Verbeugung machen: if A=20 then B=5. Vielleicht berücksichtigen wir, daß sich die meisten Computer mit kleinen Dingen nicht abgeben: **IF A=20 THEN B=5.** Und schon haben wir unsere erste BASIC-Aussage *(statement)* formuliert.
Ist doch kinderleicht, oder?

> Zusammenfassung:
> BASIC ist eine Programmiersprache, die durch sehr starke Vereinfachung aus dem Englischen hervorgegangen ist.

2.1 Satzbau

Abgesehen von den paar Vokabeln, die Sie mit den nächsten Seiten mal eben schnell überfliegen werden, erwartet der Rechner auch eine gewisse Etikette.

Was wir über die Tastatur eintippen, erscheint auf dem *Bildschirm.* Aber den Computer läßt das im übrigen kalt. Erst wenn wir die meist große und am rechten Rand der Tastatur liegende *RETURN-Taste* drücken, wird die Eingabe an den Rechner übergeben.

Grundsätzlich können die meisten Kommandos direkt eingegeben werden. Wenn wir also z. B. wissen wollen, wieviel 3+4 ist, dann formulieren wir: drucke auf den Bildschirm 3+4; kurz: drucke 3+4, englisch: **PRINT 3+4.** RETURN-Taste gedrückt, und sofort prangt eine 7 auf der Mattscheibe.

Bei fast allen Rechnern können auch mehrere Anweisungen auf einmal eingegeben werden. Dabei müssen die einzelnen Kommandos nur voneinander getrennt werden. Da der Punkt (.) schon als Markierung für die Dezimalstelle verwendet wird, muß der Doppelpunkt (:) einspringen: **PRINT 3+4:PRINT 2+5:PRINT 1+6.** RETURN-Taste gedrückt, und drei Siebenen zeigen sich auf dem Bildschirm.

Wenn ein System Mehrfachkommandos erlaubt, können so viele Befehle aufgereiht werden, wie eine *logische Zeile* faßt. Eine logische Zeile ist die Höchstmenge von Zeichen, die als eine Portion an den Rechner übergeben werden können. Bei manchen Systemen sind das zwei oder drei Bildschirmzeilen (meist je 40 Zeichen), bei anderen 256 Zeichen.

Wenn Sie nicht genau wissen, wieviele Zeichen eine logische Zeile ihres Computers faßt, dann probieren Sie es ruhig aus. Durch Herumprobieren kann nichts am Computer kaputtgemacht werden. Das Schlimmste, was geschehen kann, ist, daß der Computer »abstürzt«, d. h. er nimmt keine Eingaben mehr an. In einem solchen Fall können Sie das Gerät nur aus- und wieder einschalten oder die RESET-Taste betätigen, wenn Ihr Computer über so eine Taste verfügt.

Wenn Sie sonst etwas tun, was dem Rechner nicht angepaßt ist, dann schreibt er eine Fehlermeldung *(ERROR)* auf den Bildschirm und betrachtet seine Aufgabe als erfüllt. Noch ist es so, daß sich der Mensch (das sind Sie) der Maschine anpassen muß. Vielleicht sollten wir das ändern?

Wenn Sie längere Aufträge für Ihren Computer verfassen, dann müssen die einzelnen »Sätze« *durchnumeriert* werden. Meist sind Zeilennummern von 0 bis 65535 zulässig, manchmal auch ein paar weniger, aber selbst längere Programme bestehen nur aus einigen hundert Zeilen.

Der Rechner legt BASIC-Zeilen, die mit einer *Zeilennummer* beginnen, in seinem Speicher ab. Dabei ist es ihm ganz egal, welche Nummern die Zeilen haben und in welcher Reihenfolge sie eingegeben werden. Allerdings mag er jede Nummer nur einmal. Wird eine zweite Anweisung mit der gleichen Zeilennummer eingegeben, so geht die ältere verloren, wird überschrieben.

Später, wenn der Rechner die gespeicherten Anweisungen, das Programm, bearbeitet, beginnt er mit der kleinsten Zeilennummer und arbeitet alle Anweisungen in der Reihenfolge der Numerierung durch. Es empfiehlt sich deshalb, die Zeilennummern großzügig zu vergeben. Üblich sind *Zehnerschritte*, denn es kommt nicht selten vor, daß man im Verlauf des Programmierens feststellt, daß irgendwo noch Zeilen zwischengeschoben werden müssen.

Eine BASIC-Zeile hat immer die Form: **Zeilennummer – Anweisung: Anweisung: usw. – RETURN-Taste.** Die RETURN-Taste schreibt das Zeilenende-Zeichen (*EOL* = end of line), das auf dem Bildschirm nicht dargestellt wird.

Eine Anzahl von BASIC-Zeilen heißt BASIC-Programm und wird vom Rechner in der Reihenfolge der Zeilennummern bearbeitet.

Zusammenfassung:
Ein geschriebener Befehl wird durch Betätigung der Taste RETURN an den Rechner übergeben. Ohne Zeilennummer wird der Befehl sofort ausgeführt; mit Zeilennummer versehen, werden die Befehle im Speicher als Programm abgelegt und bei Aufruf *(RUN)* in der numerierten Reihenfolge bearbeitet.

2.2 Rechnerfutter

Rechnen kann man mit Zahlen, aber auch mit Buchstaben. Eine Zahl stellt einen bleibenden, konstanten Wert dar und wird deshalb Konstate genannt, während Buchstaben beliebige, wechselnde Werte vertreten und *Variablen* heißen.

Der Computer unterteilt ferner in konstante oder variable Werte, die in Zahlen ausdrückbar sind – die nennen wir *numerisch* – und solche Angaben, die für das menschliche Auge Text sind, für den Computer aber nichts weiter als eine bedeutungslose Folge irgendwelcher Zeichen, mit denen sein Rechenwerk absolut nichts anfangen kann. Diese Abfolge von Zeichen wird *Zeichenkette* genannt. Das englische Wort für Kette ist *String*, und das Computersymbol für String ist “$”.

Zusammenfassung:
Es wird zwischen Konstanten und Variablen unterschieden, die dem numerischen oder Stringtyp angehören können.

2.2.1 Zahlenspiele

Numerische Konstanten werden eingetippt, wie wir sie kennen, nur daß der Dezimalpunkt unser vertrautes Komma verdrängt. Allerdings hat das Fassungsvermögen eines Rechners ebensolche Grenzen wie das einer Waschmaschine.

Der Computer erlaubt nur eine begrenzte Menge von Ziffern, um einen Zahlenwert darzustellen. Werden die Zahlen so klein oder so groß, daß sie mit der begrenzten Ziffernfolge nicht mehr dargestellt werden können, dann werden sie in wissenschaftlicher Notation dargestellt. Bei dieser Form der Darstellung wird im vorderen Teil

ein Zahlenwert gegeben und danach ein Exponent, mit dem dieser Wert zu multiplizieren ist. Das sieht dann z. B. so aus:

4.6802E+06 oder **3.5791E-02,**

und das bedeutet 4.6802 mal 10 hoch 6; der vordere Zahlenwert muß also mit 1.000.000 (10 hoch 6) multipliziert werden, was ganz einfach bedeutet, daß der Dezimalpunkt um sechs Stellen nach rechts rutscht: 4680200.

Ist der Exponent negativ, so rutscht der Dezimalpunkt nach links: .035791 (eine Null vor dem Dezimalpunkt wird nicht geschrieben).

Die Codierung und Verarbeitung solcher komplizierten Zahlenausdrücke bedeutet für den Rechner allerdings eine größere Belastung, was er mit längerer Bearbeitungszeit quittiert.

Am liebsten hat er ganzzahlige Werte zwischen −32767 und +32768, die als *Intergerzahlen* bezeichnet und mit "%" gekennzeichnet werden.

Will es der Programmierer genauer wissen, läßt er den Rechner mit Realzahlen *einfacher Genauigkeit* arbeiten. Solche Werte können einen Dezimalpunkt haben und werden auf sechs Stellen genau und mit einem Exponenten (E) von −63 bis +64 dargestellt. Das Typdeklarationszeichen für Realzahlen mit einfacher Genauigkeit ist "!".

Wer ganz genaue Auskunft will, muß die *doppelte Genauigkeit* wählen. Das Zeichen dafür ist "#". Es wird auf vierzehn Stellen genau gerechnet; hinzu kommt der Exponent, der in manchen Versionen zur Unterscheidung mit "**D**" gekennzeichnet wird.

Für *numerische Variablen* gilt das gleiche. Eine Variable besteht aus einer Folge von Buchstaben und Ziffern, die immer mit einem Buchstaben beginnen muß. Meist kann der Variablenname, also die Zeichenfolge, beliebig lang sein, unterschieden werden von den meisten Rechnermodellen aber nur die ersten beiden Zeichen. Also werden die Variablennamen BA1 und BAE vom Computer so behandelt, als ob sie gleich wären.

Die Möglichkeit, längere Variablennamen zu verwenden, ist nur eine Hilfe, die Programme übersichtlicher zu gestalten. Es gibt allerdings auch Maschinen (z. B. Atari), die alle Variablennamen unterscheiden können.

Wenn der BASIC-Dialekt mit verschiedenen Typen numerischer Werte arbeitet, dann kann noch das Typdeklarationszeichen (%, !, #) hinzukommen. Eine Variable heißt dann vielleicht RB!.

Zusammenfassung:
Numerische Konstanten und Variablen können mit verschiedener Genauigkeit verarbeitet werden, die durch die Typdeklaration als ganzzahlig (%), einfachgenau (!) oder doppeltgenau (#) bestimmt werden kann.

2.2.2 Zuweisungen

Jede mögliche numerische Variable hat den Ausgangswert 0. Wenn wir also PRINT EL (RETURN) eingeben, dann erscheint eine 0 auf dem Bildschirm.

Soll eine Variable einen Wert annehmen, so müssen wir ihr einen zuweisen. Wir sagen dem Rechner: Laß GR den Wert 99 haben, kurz: LASS GR=99, englisch: LET GR=99.

LET ist also die erste Vokabel, die wir lernen. LET weist einer Variablen einen Wert zu. Das kann ein numerischer Wert sein: **LET ES=777**; das kann eine numerische Variable sein: **LET EN=S** oder es kann auch ein Ausdruck sein: **LET B=(AE+R)*BE−(L/6).**

Jede Variable behält den ihr zugewiesenen Wert, bis ihr ein anderer zugewiesen wird oder ein Befehl ergeht (z. B. NEW oder CLEAR), der alle Variablen auf Null setzt.

Weil Zuweisungen in jedem BASIC-Programm gehäuft vorkommen und man nicht nur Dutzende von LETs eintippen müßte (sie würden auch den Speicher des Rechners unnötig füllen), kann man bei den meisten BASIC-Dialekten (Ausnahme Sinclair) das LET einfach weglassen: **GR=ES*EN−S.**

Wenn Sie im Mathematikunterricht gut aufgepaßt haben, dann wissen Sie, daß bei einer Gleichung auf der rechten und auf der linken Seite vom Gleichheitszeichen der gleiche Wert stehen muß; daher der Name Gleichung. Allerdings ist es jetzt von Nachteil, das zu wissen, denn dann müssen Sie jetzt hinzulernen, daß die (LET-) Zuweisung *keine* Gleichung ist! Eine Zuweisung kann nämlich

auch lauten: VW=VW+2 – und das kann man nun wirklich nicht als Gleichung bezeichnen.

Was tut der Rechner bei dieser Zuweisung? Er nimmt den Wert der Variablen VW, addiert 2 und ordnet das Ergebnis (VW+2) der Variablen VW zu; kurz gesagt, der von der Variablen VW gehaltene numerische Wert wird um 2 erhöht.

> Zusammenfassung:
> Durch eine Zuweisung wird einer bestimmten Variablen ein bestimmter Wert zugeordnet, der gegebenenfalls den alten Wert der Variablen ersetzt. Eine Zuweisung ist keine Gleichung.

2.2.3 Operatoren

Oben haben Sie schon gesehen, daß die Rechensymbole (Operatoren) beim Computer etwas anders aussehen. Die *Multiplikation* wird durch den Asterisk (*), die *Division* durch einen Spiegelstrich (/) gekennzeichnet. Die Regel
»Punktrechnung vor Strichrechnung«
stimmt deshalb nur noch vom Sinn her, gilt aber auch für Computer.

Bestimmte Teile eines Ausdrucks können durch *Klammern* ( ) eingeschlossen werden, sie werden dann vorrangig behandelt. Der Ausdruck
2+3*4 ergibt 14, nämlich 3 mal 4 plus 2;
(2+3)*4 ergibt hingegen 20, nämlich 2 plus 3 (=5) mal 4.

Erwähnt sei noch das Zeichen für die *Potenzierung* (z. B. 4 hoch 3), die gewöhnlich mit kleinen hochgestellten Ziffern (Exponent) ausgedrückt wird, über die der Rechner aber nicht verfügt. Es wird deshalb ein Haken (∧) geschrieben und der Exponent danach in normalgroßen Ziffern.

Die umgekehrte Operation, das *Wurzelziehen*, wird auf die gleiche Weise besorgt. Man schreibt 9 hoch .5 (ein halb), was das gleiche bedeutet wie zweite Wurzel aus 9. Um die Quadratwurzel zu ermitteln, haben allerdings die meisten Rechner den Befehl SQR:
XA=SQR(9) ordnet der Variablen XA den Wert 3 (Wurzel aus 9) zu.

Außerdem werden wir noch oft *Vergleichsoperatoren* benötigen.

Das Gleichheitszeichen (=) ist einer davon, es drückt aus, daß zwei Konstanten oder Variablen gleich sind: **A=B.**

Wenn die verglichenen Werte ungleich sind, schreibt man **A<>B** oder **A><B.** Außerdem kann A kleiner sein als B: **A<B;** A kann auch kleiner oder gleich B sein: **A<=B** oder **A=<B;** A kann größer sein als B: **A>B;** und A kann größer oder gleich B sein: **A>=B** oder **A= >B.**

Zusammenfassung:
Die Operatoren werden wie aus der Mathematik bekannt verwendet, nur zum Teil in etwas abgewandelter Schreibweise.

2.2.4 Zeichenketten

Neben numerischen Werten kennt der Rechner auch Text. Folgen von Zeichen werden *String* genannt, Stringkonstanten wie wörtliche Rede in Anführungsstriche (") gesetzt, Stringvariablen durch das Typzeichen Dollar ($) gekennzeichnet.
**KA$="PAULI"** ordnet der Stringvariablen KA$ die Stringkonstante PAULI zu.

Alle Zeichen, die Ihr Heimcomputer darstellen kann – das sind Buchstaben (meist große und kleine), Ziffern und üblicherweise auch eine Anzahl Grafikzeichen –, können in einem String vorkommen, nur eben kein Anführungszeichen, denn das markiert ja Anfang beziehungsweise Ende des String.

Intern wandelt der Rechner die Schriftzeichen (engl.: characters) natürlich in Zahlenwerte um. Dafür gibt es zum Glück einen internationalen Standard aus der Fernmeldetechnik, den *ASCII-Code.* Das ist die Abkürzung für American Standard Code for Information Interchange, was auf gut deutsch »amerikanische Norm für Datenübertragung« heißt.

Der ASCII-Code ordnet jedem Zeichen eine Zahl zwischen 0 und 127 zu. Bei Heimcomputern ist das inzwischen auf Werte bis 255 erweitert worden. Wenn Sie den Code eines Zeichens wissen wollen, dann befehlen Sie Ihrem Computer: **PRINT ASC(**"Zeichen").

Wenn Sie als Zeichen »A« einsetzen, dann bekommen Sie 65 ausgegeben. Auch für den umgekehrten Vorgang gibt es ein BASIC-Kommando. Es lautet **CHR$** (CHaRacter):

PRINT CHR$ (65) schreibt ein A auf die Mattscheibe. Und mit diesem schönen Wissen können wir nun auch ein Anführungszeichen (ASCII 34) einer Stringvariablen zuordnen: **AN$=CHR$(34):PRINT AN$**

Mit Strings können alle Vergleichsoperationen ausgeführt werden. Ein Zeichen ist größer als ein anderes, wenn sein Zeichencode (ASCII-Wert) größer ist usw.

Rechnen kann man mit Strings natürlich nicht. Aber in den meisten BASIC-Dialekten kann man Strings *verketten*, indem man sie addiert:
**A$="123":B$="456":C$=A$+B$** ordnet C$ die Zeichenfolge »123456« zu.

Zusammenfassung:
Strings sind für den Rechner bedeutungslose Folgen von Zeichen. Stringvariablen werden durch $ gekennzeichnet, Stringkonstanten in " " eingeschlossen.

2.2.5 Felder

Eine nützliche Einrichtung bietet die Möglichkeit, ganze Batterien von Variablen zu benennen. Man spricht von *Feldvariablen* (Arrays).

Während eine einfache Variable einer Schublade gleicht, auf deren Etikett der Variablenname steht, sind Feldvariablen wie eine Regalwand voller Schubladen.

Eine Feldvariable muß *dimensioniert* werden. Dafür gibt es den BASIC-Befehl **DIM:**

DIM A(9) richtet eine Regalwand mit zehn (0 bis 9) Schubladen ein. Die einzelnen Variablen heißen dann A(0), A(1), A(2)... bis A(9). Das Praktische an diesen Feldvariablen besteht auch darin, daß der Index (das ist die Zahl in der Klammer) durch eine Variable ersetzt werden kann:

M=5:A(M)=45:PRINT A(5) führt zur Ausgabe der Zahl 45 auf dem Bildschirm.

Arrays können *mehrfach* dimensioniert werden. Zum Beispiel richtet DIM FG(7,7) 64 Variablen mit dem Namen FG (n,n) ein. Ein-

zelne BASIC-Dialekte erlauben Mehrfachdimensionierungen bis hin zu 255 Dimensionen (z. B. bei MSX).

Üblicherweise können auch Stringvariablen dimensioniert werden: DIM TX$(3,9,12). Aber es gibt Ausnahmen (Atari), die keine dimensionierten Stringvariablen zulassen.

Feldvariablen belegen sehr viel Platz im Speicher. Man sollte deshalb nur so groß dimensionieren, wie es *tatsächlich* für das Programm benötigt wird.

Zusammenfassung:
Durch Dimensionierung können Variablengruppen geschaffen werden, deren Elemente durch einen in Klammern gesetzten Index unterschieden werden. Der Index ist Teil des Variablennamens.

2.3 Grundbefehle

Sie werden nicht schon ungeduldig? Bevor wir so richtig mit dem Programmieren loslegen können, sollten Sie die vier Grundbefehle kennenlernen, die das Rückgrat jedes BASIC-Programms bilden.

Einer dieser Befehle dient der Ausgabe auf dem Bildschirm, die übrigen drei bilden die *Struktur* des Programms, in der alle anderen BASIC-Kommandos, manche Dialekte bestehen aus bis zu 150 Einzelbefehlen, nur einen untergeordneten Platz einnehmen.

2.3.1 Ausgabe

Der Ausgabebefehl wird so häufig gebraucht, daß es nicht zu vermeiden war, ihn weiter oben schon zu verwenden. **PRINT**, das fast immer durch ? ersetzt werden kann, veranlaßt die Ausgabe auf dem Bildschirm oder Monitor.

Wenn nichts anderes bestimmt wird, beginnt die Darstellung am linken oberen Bildschirmrand, erfolgt fortlaufend nach rechts und wird gegebenenfalls in der nächsten Zeile fortgesetzt; abschließend springt der Läufer (Cursor) an den Anfang der nächsten Zeile.

PRINT 13 führt zu einer Ausgabe der numerischen Konstanten 13 auf dem Bildschirm.

Wird hinter PRINT ein Ausdruck gegeben, so wird der numerische Wert des Ausdruckes (das Ergebnis) in dezimaler Form ausgedruckt:

PRINT 3*5 – Bildschirmausgabe: 15.

Wird eine numerische Variable für die Ausgabe bestimmt, so wird ihr Inhalt dargestellt:

PRINT A ergibt die Ausgabe von 0, wenn A nicht zuvor ein Wert zugeordnet wurde.

Mit PRINT kann auch die Ausgabe von Strings bestimmt werden:

PRINT "A" druckt ein A auf den Bildschirm,

**PRINT D$** bewirkt die Ausgabe eines Leerstrings (" "), was auf dem Bildschirm ohne sichtbare Wirkung bleibt. Wurde D$ zuvor ein Wert zugeordnet, D$="KATER PAULI", so wird dieser Wert durch PRINT zur Ausgabe gebracht.

Mit *einem* PRINT-Kommando können *mehrere* Werte zur Bildschirmdarstellung angeordnet werden:

20 PRINT A;" ";B$,"te"
Wenn zuvor den verwendeten Variablen Werte zugeordnet wurden:

10 A=765:B$="vier" schreibt das Programm folgende Zeile auf den Bildschirm:

765 vier te

Das *Semikolon* (;) nach einer PRINT-Anweisung bewirkt, daß der Cursor hinter dem Ausdruck stehenbleibt, statt an den Anfang der nächsten Zeile zu springen. Die nächste Ausgabe erfolgt also *direkt hinter* der vorhergehenden. In unserem Beispiel wird direkt hinter dem numerischen Wert der Variablen A (765) die aus zwei Leerstellen bestehende Stringkonstante ausgedruckt und wieder direkt danach der Wert von B$.

Das *Komma* (,) hinter B$ bewirkt, daß der Cursor zur nächsten *Tabulatormarke* springt. Welche Tabulatormarken bei Einschalten des Computers vorgegeben sind, ist von Gerät zu Gerät verschieden. Gelöscht und gesetzt werden die Tabulatormarken in der Regel durch entsprechende Tasten, ähnlich wie bei einer elektrischen Schreibmaschine.

Um bestimmte Stellen des Bildschirms zu erreichen, verfügen fast

alle BASIC-Dialekte über einen Befehl, der den Cursor auf eine angegebene *Position* stellt.

Die *Schreibstellen* des Bildschirms werden dazu durch *Koordinaten* benannt. Fast alle Heimcomputer schreiben 40 Zeichen pro Zeile über 24 Zeilen; die Spaltenpositionen (X-Achse) reichen von 0 bis 39, die Zeilenpositionen (Y-Achse) von 0 bis 23. Die Schreibposition in der linken oberen Ecke des Bildschirms hat die Koordinaten (0,0), die Ecke unten rechts (39,23). Etwas Verwirrung entsteht dadurch, daß einige Computermodelle (z. B. Sinclair) die Koordinaten in den Reihenfolgen (Y,X) erwarten, während die meisten Rechner der mathematischen Gepflogenheit (X,Y) folgen.

Die Befehle, den Cursor an eine bestimmte Stelle zu stellen, lauten je nach BASIC-Version z. B. **PRINT AT, LOCATE** oder **POSITION:**

```
10 A$="HIER"
20 LOCATE 20,5
30 PRINT A$; "UND";
```

setzt den Cursor in Zeile 5 auf Spalte 20, druckt, dort beginnend, den String "HIER", gleich anschließend "UND" und hält den Cursor dahinter an.

> Zusammenfassung:
> PRINT veranlaßt die Ausgabe auf dem Monitor, PRINT kann durch ? ersetzt werden. Es gibt Kommandos, um für die Ausgabe eine bestimmte Position auf dem Bildschirm vorzugeben.

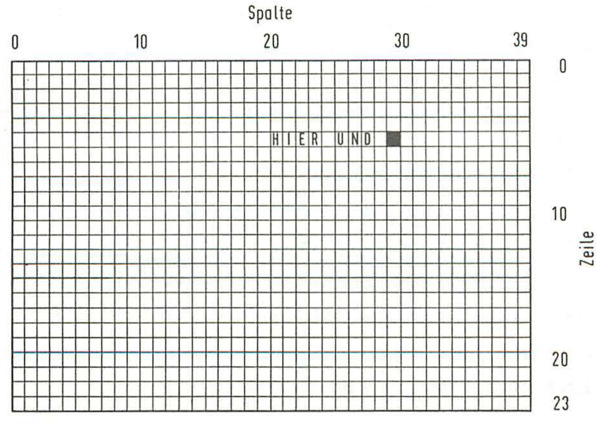


Abb. 1: Der Ausdruck beginnt in Spalte 20 von Zeile 5

2.3.2 Sprünge

Wie Sie inzwischen wissen, werden die Zeilen eines BASIC-Programms der Numerierung folgend bearbeitet. Mit einem *Sprungbefehl* kann man bewirken, daß die Programmausführung bei einer bestimmten Zeile fortgesetzt wird:

20 GOTO 10

Der Sprungbefehl wird sofort ausgeführt, d. h. in der gleichen Programmzeile können hinter GOTO (dtsch.: gehe nach) keine weiteren Anweisungen folgen. In 30 GOTO 10:PRINT"F" wird das "F" nicht ausgegeben, da vorher zur Zeile 10 gesprungen wird.

Mit Hilfe von GOTO können unendliche Programme geschrieben werden:

```
10 A=A+1
20 PRINT A
30 GOTO 10
```

veranlaßt den Rechner, bei 1 beginnend zu zählen und die Zahlen auf dem Bildschirm zu zeigen. Durch den in Zeile 30 verordneten Rücksprung nach Zeile 10 läuft das Programm endlos weiter, bis eine Unterbrechungstaste (z. B. STOP) gedrückt wird.

Einige BASIC-Versionen (z. B. Atari) lassen Variablen und Ausdrücke als Sprungziele zu. Folgende Anweisung ist dann möglich:

150 GOTO A*100

Natürlich muß dafür gesorgt sein, daß alle möglichen Zeilennummern, die der Ausdruck A*100 darstellen kann, auch im Programm vorkommen. Bei den meisten BASIC-Dialekten können nur *tatsächliche* Zeilennummern als Sprungziel bestimmt werden; dort gibt es aber in der Regel Kommandos für einen bedingten Sprung:

170 ON SZ GOTO 100, 200, 300, 350

Dieser Befehl bewirkt, daß der Sprung in Abhängigkeit der Variablen SZ erfolgt. Hat SZ den Wert 1, erfolgt der Sprung zu der ersten hinter GOTO bestimmten Zeilennummer; hat SZ den Wert 2, gibt die zweite Zeilennummer das Sprungziel an und so fort.

Oft sind auch andere Bedingungen für den Sprung möglich, z. B. **ON ERROR GOTO, ON KEY GOTO** und ähnliche.

Eine Variante stellt der Sprung in ein Unterprogramm **(GOSUB)**

dar. Das Besondere daran ist, daß ein Unterprogramm strukturell vom Hauptprogramm getrennt ist und immer mit einem Rücksprung (**RETURN**) abschließt:

```
10 Hauptprogramm
20 Hauptprogramm
30 Hauptprogramm
40 GOSUB 1000
50 Hauptprogramm
60 Hauptprogramm
70 Hauptprogramm
80 Hauptprogramm
90 END
1000 Unterprogramm
1010 Unterprogramm
1030 RETURN
```

Das Hauptprogramm muß mit **END** gegen das Unterprogramm abgegrenzt sein, da sonst die Programmbearbeitung nach Zeile 90 mit dem Unterprogramm (Zeile 1000) fortgesetzt würde.

In Zeile 40 wird der Sprung nach Zeile 1000 angeordnet. Dort wird das Unterprogramm abgearbeitet und schließlich mit RETURN der Rücksprung veranlaßt, der die Fortsetzung der Bearbeitung in der Zeile bewirkt, die der Zeile mit dem GOSUB-Statement folgt; das ist in diesem Fall Zeile 50.

Bei manchen BASIC-Dialekten ist es möglich, zusammen mit RETURN eine Zeilennummer für den Rücksprung zu bestimmen.

Jedes Unterprogramm kann *beliebig oft* angesprungen werden. Jedes Programm kann *beliebig viele* Unterprogramme enthalten. Wichtig ist nur, daß ein Unterprogramm immer mit RETURN verlassen wird, weil der Rechner sonst mit den Rücksprung-Zeilennummern durcheinanderkommt und das Programm nicht die gewünschte Wirkung erbringen wird.

Ähnlich wie bei GOTO bieten viele BASIC-Versionen auch Kommandos der Form "**ON** Bedingung **GOSUB** Zeilennummer".

Zusammenfassung:
Mit GOTO kann eine Zeilennummer bestimmt werden, bei der die Bearbeitung des Programms fortgesetzt werden soll.

2.3.3 Bedingungen

Die zweite Möglichkeit, ein Programm zu strukturieren, besteht darin, Anweisungen von Bedingungen abhängig zu machen. Ein solcher Befehl hat die Form »WENN Bedingung DANN Anweisung«, auf englisch: **IF... THEN.**

Die Bedingung ist immer eine Vergleichsoperation; die Anweisung kann ein Sprung (GOTO/GOSUB), eine Zuordnung (LET), eine Ausgabe (z. B. PRINT) oder, bei manchen Versionen, auch eine weitere IF-Bedingung sein.

```
10 IF A=B THEN GOTO 100
```

Nur wenn die Variablen A und B den gleichen Wert haben, erfolgt ein Sprung nach Zeile 100. Bei fast allen BASIC-Dialekten kann man eines der beiden Wörter, THEN oder GOTO, fortlassen:

```
20 IF A<B THEN 200
```

oder

```
30 IF A<>B GOTO 300
```

Wenn in der BASIC-Zeile hinter der IF-Bedingung mehrere Befehle aufgereiht sind, dann werden die hinteren Anweisungen nur ausgeführt, wenn die Bedingung erfüllt ist:

```
40 IF A>B THEN C=5:B=6
```

Wenn A größer als B ist, bekommt C den Wert 5, und nur dann bekommt auch B den Wert 6. Ist die Bedingung nicht erfüllt, A kleiner als B, wird die Programmbearbeitung sofort bei der nächsten Zeilennummer fortgesetzt und auch die Anweisung B=6 nicht bearbeitet.

Anspruchsvollere BASIC-Versionen erlauben zu IF eine Alternative, die durch »SONST« bezeichnet wird: WENN Bedingung, DANN Anweisung, SONST Anweisung; engl.: **IF** Bedingung **THEN** Anweisung **ELSE** Anweisung:

```
50 IF A=B THEN C=5 ELSE C=6
```

Haben A und B den gleichen numerischen Wert, wird die Variable C auf 5 gesetzt; sind A und B nicht gleich, bekommt C den Wert 6.

Ein kleines Spiel zur Entspannung:

Der Computer denkt sich eine Zahl zwischen 0 und 999, und der Benutzer soll diese Zahl mit möglichst wenigen Versuchen heraus-

finden. Natürlich kann sich kein Rechner etwas ausdenken; aber BASIC hat eine Funktion, um einen *zufälligen Zahlenwert* zu finden. Diese Funktion wird mit dem BASIC-Wort **RND,** engl. 'random' (beliebig), aufgerufen.

```
0 REM ZAHLRATE.ATA
100 RZ=INT(RND(0)*1000)
200 ? CHR$(125):? "Raten Sie eine Zahl":? :?
"zwischen 0 und 999:"
210 V=V+1:? :? "Ihre Zahl-------->";:INPUT ZU
220 IF ZU<RZ THEN ? :? "zu klein!"
230 IF ZU>RZ THEN ? :? "zu gross!"
240 IF ZU=RZ THEN ? :? "richtig geraten mit n
ur ";V;" Versuchen!":END
250 GOTO 210
```

100: In der Variablen RZ wird die zufällig ausgewählte Zahl aufgehoben. Die Funktion RND findet eine Zahl, die größer oder gleich Null und kleiner als 1 ist. Da wir Zahlen von 0 bis 999 haben wollen, multiplizieren wir den RND-Wert mit 1000. Findet RND eine Null, dann ist 0*1000=0; der größtmögliche Wert für RND ist 0,99..., der mit 1000 multipliziert den Wert 999,99... ergibt. Da die Ratezahl eine ganze oder Integerzahl sein soll, unterdrücken wir mit der Funktion **INT** (integer) die Stellen hinter dem Komma.

200: Bei Atari entspricht das Zeichen 125 dem Bildschirmlöschen. ; PRINT CHR$(125) löscht also den Bildschirm und stellt den Cursor in Home-Position, das ist die Ecke oben links.

210: Die Variable V zählt die Anzahl der Rateversuche. Bei jedem Durchlauf wird der Wert von V um 1 erhöht. Der INPUT-(Eingabe)Befehl unterbricht die Programmbearbeitung. Auf dem Bildschirm erscheint ein Fragezeichen. Der Benutzer soll über die Tastatur etwas eingeben. In unserem Fall die Zahl, auf die er tippt. Die Benutzereingabe muß mit der Taste RETURN beendet werden. Die eingegebene Zahl wird der hinter INPUT bestimmten Variablen ZU zugeordnet; dann wird die Programmbearbeitung fortgesetzt.

Natürlich denkt ein professioneller Programmierer daran, daß ein Benutzer auch *aus Versehen* z. B. eine Buchstabentaste bei der Eingabe drücken kann. Probieren Sie aus, was passiert. Eine Buchstabentaste würde eine Stringeingabe bedeuten. Aber der numerischen

Variablen kann keine Stringkonstante zugeordnet werden. Was soll der Rechner tun? Er macht es sich einfach, bricht die Programmbearbeitung ab und gibt eine *Fehlermeldung* aus. Ein Programm gegen Fehleingaben abzuschotten, ist meist ein so aufwendiges Unterfangen, daß wir uns an dieser Stelle noch nicht damit auseinandersetzen wollen.

220 bis 240: Jetzt wird die mit INPUT aufgenommene Zahl des Benutzers mit der zu ratenden Computerzahl verglichen und in Abhängigkeit (IF) von dieser Vergleichsoperation ein entsprechendes Ergebnis ausgegeben. Sind beide Zahlen gleich, ENDet das Programm; sonst wird es in Zeile 250 nach Zeile 210 zurückgeführt.

Atari holt die Daten für die RND-Funktion aus einem Hardware-Baustein des Tongenerators. Die so erzeugten Zufallszahlen sind echte Zufallszahlen. Fast alle anderen Heimcomputer arbeiten mit einer arithmetischen Funktion, was dazu führt, daß immer die gleiche Reihenfolge von Zufallszahlen gefunden wird. Man spricht deshalb von »Pseudozufallszahlen«.

Um trotzdem bei jedem Programmlauf mit RND andere Zahlen zu finden, kann irgendwo in dieser immer gleichen Zahlenreihe begonnen werden. Wird in der Klammer hinter RND ein negativer Wert bestimmt, so setzt er einen Ausgangswert für die Zahlenreihe. Natürlich ist die danach erzeugte Abfolge von Zufallszahlen auch jedesmal gleich. Deswegen wird noch ein wechselnder Wert für den RND-Start benötigt.

Nun verfügt jeder Computer über eine interne *Uhr*. Das sind mehrere Speicherzellen, in denen fortlaufend gezählt wird. Die Zählimpulse liefert der Hertz-Schlag des Wechselstroms.

Diese interne Uhr beginnt beim Einschalten des Rechners zu zählen. Da es jedesmal eine andere Zeitspanne vom Einschalten des Systems bis zum Programmstart dauern wird, enthält die interne Uhr jedesmal einen anderen Wert, zumal sie in *einer Sekunde 60 Takte* zählt.

Beim MSX-BASIC enthält die Variable TIME den aktuellen Wert des internen Taktgebers, RND(-TIME) wird also jedesmal einen anderen Anfangswert für die Reihe der Pseudozufallszahlen setzen. Etwas umständlich, aber Sie werden sich daran gewöhnen:

```
0  REM ZAHLRATE.MSX
100 RN=RND(-TIME)
110 RZ=INT(RND(1)*1000)
200 CLS:PRINT"Raten Sie eine Zahl":PRINT:PRIN
T"zwischen 0 und 999:"
210 V=V+1:PRINT:INPUT"Ihre Zahl------->";ZU
220 IF ZU<RZ THEN PRINT:PRINT"zu klein!"
230 IF ZU>RZ THEN PRINT:PRINT"zu groß!"
240 IF ZU=RZ THEN PRINT:PRINT"richtig geraten
 mit nur";V;"Versuchen!":END
250 GOTO 210
```

100: Die Variable RN hat überhaupt keine Bedeutung, die Funktion RND(-TIME) setzt nur den RND-Anfangswert, und die Syntax erfordert die Zuordnung zu einer Variablen.

110: Hier wird dann die eigentliche Ratezahl gefunden.

200: MSX-BASIC verfügt über einen Befehl zum Bildschirmlöschen: **CLS.**

In den Grundfunktionen sind alle BASIC-Dialekte sehr ähnlich. Deshalb gibt es auch in der Commodore-Version dieses kleinen Programms kaum Unterschiede.

```
0 REM ZAHLRATE.COM
10 R=RND(-TI)
100 RZ=INT(RND(1)*1000)
200 PRINTCHR$(147);"RATEN SIE EINE ZAHL":PRIN
T:PRINT"ZWISCHEN 0 UND 999:"
210 V=V+1:PRINT:INPUT"IHRE ZAHL------>";ZU
220 IF ZU<RZ THEN PRINT:PRINT"ZU KLEIN!"
230 IF ZU>RZ THEN PRINT:PRINT"ZU GROSS!"
240 IF ZU=RZ THEN PRINT:PRINT"RICHTIG GERATEN
 MIT NUR";V;"VERSUCHEN!":END
250 GOTO 210
```

10: Die Timer-Variable heißt hier nicht TIME, wie bei MSX, sondern nur **TI,**
200: und der Bildschirm wird mit CHR$(147) gelöscht.

> Zusammenfassung:
> Die Ausführung von Befehlen kann von einer Bedingung abhängig gemacht werden. Wenn die Bedingung (IF) nicht erfüllt ist, wird die Bearbeitung des Programms in der nächsten Zeile fortgesetzt.

2.3.4 Schleifen

Wie jede andere Maschine hat der Rechner seine Stärke in der stupiden Routinearbeit. Deshalb ist ein Befehl ganz besonders wichtig, der es ermöglicht, mit wenigen Anweisungen sehr viele *gleiche Vorgänge* auszulösen. Durch eine Schleife wird ein Programmteil mehrmals abgearbeitet. Der Schleifenbefehl kontrolliert dabei die *Anzahl* der Durchläufe.

Eine beliebige numerische Variable wird als Schleifenzähler verwendet. Bei jedem neuerlichen Durchlauf wird die Schleifenvariable weitergezählt. Außerdem bestimmen wir einen Endwert für den Schleifenzähler und postieren einen »Aufpasser« am Ende der Schleife.

Soll die Schleife zehnmal durchlaufen werden, so kann der Befehl lauten: Für J von 0 bis 9, kurz: FÜR J=0 BIS 9, engl.: **FOR** J=0 **TO** 9; und am Ende der Schleife postieren wir den Befehl NÄCHSTES J, engl.: **NEXT J.**

```
10 FOR H=4 TO 8
20 PRINT CHR$(65);
30 NEXT H
```

Das Programm druckt fünfmal den Buchstaben A auf den Monitor, und zwar so:

In Zeile 10 bekommt H den Wert 4; in Zeile 20 wird das Zeichen 65 (A) ausgegeben; und in Zeile 30 überprüft NEXT den augenblicklichen Wert von H mit dem bestimmten Endwert. Da H zur Zeit den Wert 4 hält, der Endwert aber auf 8 gesetzt ist, erfolgt der Rücksprung zur Zeile 10. Erst wenn H beim fünften Durchlauf den Wert 8 hält, gibt NEXT den Weg frei.

Eine vollständige FOR...NEXT-Schleife kann in einer einzigen BASIC-Zeile stehen:
10 FOR W=0 TO 99: PRINT CHR$(7):NEXT W läßt hundertmal einen Signalton (ASCII-Code 7) ertönen, bevor das Programm weiterläuft.

10 FOR P=0 TO 10000:NEXT P bewirkt, daß in der Programmbearbeitung eine *Pause* eintritt, denn der Rechner wird veranlaßt von 0 bis 10000 zu zählen, was er zwar recht flink tut, was aber trotzdem einen Moment dauert.

Natürlich kann mit dem Schleifenzähler auch innerhalb der Schleife operiert werden:

```
10 FOR K=3 TO 12
20 PRINT K*K
30 NEXT K
```

bringt die Quadratzahlen (K*K) von 9 (3*3) bis 144 (12*12) auf den Bildschirm.

Nun kann der Schleifenzähler aber nicht nur jeweils um 1 hochgezählt werden. Mit der Ergänzung STEP (Schritt) kann auch bestimmt werden, *wie* gezählt werden soll; z. B.:

100 FOR R=–4 TO 4 STEP .5

oder

110 FOR S=10 TO 0 STEP –.3

Ein Programm kann beliebig viele Schleifen enthalten, und innerhalb einer Schleife können andere Schleifen bestimmt werden; die Schleifen dürfen sich nur nicht kreuzen, d. h. die *zuletzt* begonnene Schleife muß auch *als erste* beschlossen werden:

```
10 FOR A=0 TO 9
20 FOR B=3 TO 4 STEP .1
30 FOR C=1 TO 3
40 NEXT C
50 NEXT B
60 FOR D=0 TO 5
70 NEXT D
80 NEXT A
```

In der Praxis werden so vielfach verschachtelte Schleifen wohl seltener vorkommen. Aber um einer doppeltindizierten Variablen Werte zuzuweisen, kann eine Schleife sehr nützlich sein.

```
10 FOR I=0 TO 9
20 FOR J=0 TO 5
30 M(I,J)=3
40 NEXT J
50 NEXT I
```

Allen sechzig Variablen M (n,n) wird der Wert 3 zugeordnet. Auch diese Anweisung ließe sich in einer Zeile schreiben:

10 FOR I=0 TO 9:FOR J=0 TO 5:M(I,J)=3:NEXT J: NEXT I

Anfangs- und Endwerte für den Schleifenzähler können auch durch Variablen und Ausdrücke bestimmt werden:

10 W=5:V=3
20 FOR D=V+2 TO V*W STEP W

Im folgenden kleinen Programm wird mit Hilfe einer FOR-...NEXT-Schleife der Bildschirm mit Buchstaben vollgeschrieben. Wenn Sie Lust haben, können Sie ein kleines Spiel daraus machen, in dem zufälligen Letternsalat sinnvolle Wörter zu suchen. Bewegen Sie den Cursor hoch, und löschen Sie alle Buchstaben um das gefundene Wort herum, so daß Ihre Entdeckung sauber zu lesen ist. Wer findet die meisten und längsten Wörter?

```
0 REM FORNEXT.COM
10 PRINT CHR$(147)
20 FOR ZE=0 TO 22
30 FOR SP=0 TO 39
40 REM
50 X=INT(RND(7)*26)+65
60 PRINT CHR$(X);
70 NEXT SP
80 NEXT ZE
```

10 löscht den Bildschirm.

20: ZE zählt 24 Zeilen.

30: SP zählt 40 Spalten.

40: Alles, was hinter **REM** (von engl.: remark = Bemerkung) steht, wird nicht vom Rechner bearbeitet. Auf diese Weise können in einem BASIC-Programm Anmerkungen untergebracht werden, die zur Orientierung beim Programmieren beitragen. Allerdings belasten REM-Bemerkungen den Speicher. Sie benötigen Platz und sind doch ohne Wirkung. REM-Zeilen sollten deshalb in der Regel entfernt werden, wenn ein Programm geschrieben ist und fehlerfrei läuft. Hier steht eine REM-Zeile, weil in den Versionen für Atari und Commodore in dieser Zeile eine wirksame Anweisung steht.

50: Es sollen *per Zufall* Buchstaben ausgedruckt werden. Die ASCII-Codes der Großbuchstaben von A bis Z lauten 65 bis 90. Wir benötigen also Zufallszahlen in einer Breite von 0 bis 25, zu denen wir dann den Wert 65 addieren, um schließlich auf Werte von 65 bis 90 zu kommen.

60: Unter Beihilfe des zufälligen Wertes und der CHR$-Funktion wird ein Buchstabe auf dem Bildschirm dargestellt.

Bei Commodore können die Zeichen nur *einfach fortlaufend* ausgegeben werden, wodurch natürlich der ganze Bildschirm vollgeschrieben wird. Dazu hätte aber auch eine einfache Schleife genügt, die von 0 bis 999 läuft, denn der Commodore-Bildschirm faßt tausend Zeichen.

In der Atari-Version kann der Cursor mit dem **POSITION**-Befehl auf eine *bestimmte Stelle* des Bildschirms gesetzt werden. Bei diesem kleinen Beispiel ist die Wirkung allerdings die gleiche wie beim fortlaufenden Commodore-Ausdruck:

```
0 REM FORNEXT.ATA
10 PRINT CHR$(125):POKE 82,0
20 FOR ZE=0 TO 20
30 FOR SP=0 TO 39
40 POSITION SP,ZE
50 X=INT(RND(0)*26)+65
60 ? CHR$(X);
70 NEXT SP
80 NEXT ZE
```

Für MSX sieht das Programm ganz ähnlich aus; nur heißt der Positionierungsbefehl hier **LOCATE**:

```
0  REM FORNEXT.MSX
10 CLS:WIDTH 40:KEY OFF
20 FOR ZE=0 TO 20
30 FOR SP=0 TO 39
40 LOCATE SP,ZE
50 X=INT(RND(1)*26)+65
60 PRINT CHR$(X);
70 NEXT SP
80 NEXT ZE
```

Zusammenfassung:
Mit FOR...NEXT können Programmabschnitte mehrfach bearbeitet werden. Mit FOR wird eine Variable zum Schleifenzähler bestimmt. Ohne Angabe einer Schrittweite durch **STEP** wird der Schleifenzähler bei jedem Durchlauf um 1 hochgezählt.Das Ende der Schleife wird durch NEXT markiert. Hat der Schleifenzähler noch nicht den hinter TO bestimmten Endwert erreicht, erfolgt Rücksprung zu der Zeile, in der das entsprechende FOR programmiert ist. Beliebig viele FOR...NEXT-Schleifen können verschachtelt sein, dürfen sich aber niemals überkreuzen.

2.3.5 Sortieren

Nachdem Sie nun die Grundbefehle und einige Varianten dazu kennen, möchte ich Ihnen zum Abschluß dieses Kapitels eine einfache Programmiertechnik vorstellen, mit der Daten sortiert werden können.

Das Programm erzeugt zuerst per Zufall zwanzig Zahlen und sortiert diese dann in aufsteigender Reihe.

```
0  REM SORTZAHL.MSX
10 DIM Z(19)
20 R=RND(-TIME)
30 FOR J=0 TO 19
40 Z(J)=INT(RND(1)*1000)
50 NEXT
60 CLS:FOR J=0 TO 19:PRINT USING"####";Z(J):N
EXT
100 K=0
110 FOR J=0 TO 18
120 IF Z(J)>Z(J+1) THEN SWAP Z(J),Z(J+1):K=1
130 NEXT
140 IF K<>0 THEN 100
200 FOR J=0 TO 19:LOCATE 9,J:PRINT USING"####
";Z(J):NEXT
```

10: Die zwanzig Zufallszahlen werden in einer indizierten Variablen erfaßt, denn durch den Index können die zwanzig Variablen geordnet werden. Indizierte Variablen werden auch Feldvariablen oder Arrays (engl.: Ordnung, Reihe) genannt. Bei MSX muß eine Feldvariable nur dimensioniert werden, wenn sie mehr als zehn Elemente haben soll. Das ist hier gegeben. Wir DIMensionieren die Variable Z(n) auf zwanzig (0 bis 19).

20 setzt den Anfangswert für die Pseudozufallszahlen.

30: Der Schleifenzähler J soll von 0 bis 19 laufen; in Zeile

40 wird J als Index für die Feldvariable verwendet. Beim ersten Durchlauf der FOR...NEXT-Schleife wird also Z (0) bearbeitet, beim zweiten Z(1) usw.

50: Da der Bezug eindeutig ist, muß mit NEXT nicht der bezogene Schleifenzähler (hier J) genannt werden.

60: Jetzt sind in Z(n) zwanzig Zufallswerte von 0 bis 999 in willkürlicher Ordnung abgelegt. Diese Programmzeile zeigt uns die

Werte auf der Mattscheibe. **PRINT USING "####"** bestimmt, daß jeder Zahlenwert so behandelt wird, als ob er vierstellig wäre. Dadurch stehen alle ausgegebenen Zahlen sauber untereinander.

100: Jetzt sollen die zwanzig Zahlen in aufsteigender Folge sortiert werden. Aber wie?
Ein Computer ist dumm. Wir können ihm nicht erklären, was Sortieren bedeutet. So einfach es klingen mag, es ist ein recht komplizierter Vorgang. Versuchen Sie einmal zu beschreiben, wie man etwas sortiert!
Der Computer ist dumm, aber schnell. Wir können ihn die Arbeit sehr umständlich machen lassen, und er wird doch noch schneller fertig sein als wir. Ein umständlicher Sortiervorgang ist aber einfacher zu beschreiben.
Stellen Sie sich die zwanzig Zahlen in einer Reihe vor, wie sie ganz willkürlich angeordnet nebeneinander stehen. Wir hätten nun gerne, daß der kleinste Wert ganz links stände und die übrigen Werte aufsteigend folgten, bis rechts außen die höchste Zahl den Abschluß bilden würde.
Betrachten wir jeweils nur zwei nebeneinanderstehende Werte, so ist diese Forderung leicht zu erfüllen: Ist die linke Zahl kleiner als die rechte, dann ist alles in Ordnung; ist die linke größer, dann müssen beide ihren Platz tauschen. Diese Anweisung ist so einfach, daß sie sogar einem Computer eingegeben werden kann. In Zeile

120 ist dieser Befehl formuliert.

Wenn der Wert der Z-Variablen mit dem kleineren Index größer ist als der Wert der Z-Variablen mit dem nächsthöheren Index, dann sollen die beiden Werte ausgetauscht werden. MSX hält für das Austauschen den Befehl **SWAP** (vertauschen) bereit.

110 und 130: Diese einfache Tauschanweisung wird nun die ganze Reihe von zwanzig Zahlen für jedes einzelne der neunzehn Paare durchgegangen. Wenn das erledigt, wenn die FOR...NEXT-Schleife durchgelaufen ist, dann ist die Ordnung in der Reihe sicherlich größer. Vielleicht ist sie dann auch schon perfekt. Aber woher sollen wir das wissen; nein, richtiger: Wie soll der Rechner das wissen? Wie sagen wir ihm, wann die von uns gewünschte Ordnung erreicht ist und wann er noch einen weiteren Sortiergang durchführen muß?

Nun, die Lösung ist ganz einfach. Wenn die FOR...NEXT-Schleife bearbeitet wird und keine einzige Vertauschung notwendig war, dann ist unsere Ordnung erreicht. Deshalb wird in Zeile 120 die Variable K zur Kontrolle auf 1 gesetzt, wenn (IF) eine Vertauschung vorgenommen wurde.

140: Wenn auch nur eine einzige Vertauschung notwendig war, dann mag es sein, daß die Ordnung noch nicht fertig ist; und wir führen das Programm mit einem Rücksprung (GOTO) noch einmal durch die FOR...NEXT-Schleife der Zeilen 110 bis 130. Wenn also K ungleich 0 ist, muß noch einmal sortiert werden. Das Programm wird zur Zeile 100 geführt. K bekommt hier den Wert Null, und ein weiterer Sortiervorgang läuft ab.
Wenn K in Zeile 140 noch immer den Wert 0 hat, dann war keine Umstellung von Zahlen mehr nötig; die Ordnung ist fertig, die Bedingung (IF) in dieser Zeile nicht erfüllt: Die Bearbeitung wird mit Zeile

200 fortgesetzt, wo die sortierten Zahlen in einer zweiten Spalte auf den Bildschirm geschrieben werden. Der Befehl LOCATE stellt den Cursor auf die bestimmte Position des Bildschrims. Der erste Wert setzt die Spalte (X-Achse), der zweite die Zeile (Z-Achse).

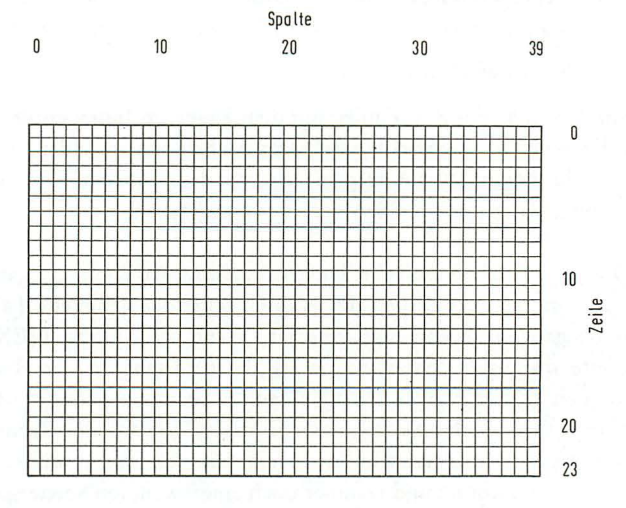


Abb. 2: Aufteilung des Bildschirms in 40 Spalten und 24 Zeilen

In diesem Fall beginnt der Ausdruck in Spalte 9, während die Zeile von J abhängt, d. h. die Feldvariable Z(J) wird in Zeile J auf die Mattscheibe geschrieben.

Für diesen kompliziert klingenden Sortiervorgang benötigt der Rechner übrigens nur wenige Sekunden.

Das gleiche Programm sieht für Atari ganz ähnlich aus:

```
0 REM SORTZAHL.ATA
10 DIM Z(19)
20 FOR J=0 TO 19
30 Z(J)=INT(RND(0)*1000)
40 NEXT J
50 ? CHR$(125)
60 FOR J=0 TO 19:POSITION 2,J:? Z(J);:NEXT J
100 K=0
110 FOR J=0 TO 18
120 IF Z(J)>Z(J+1) THEN U=Z(J):Z(J)=Z(J+1):Z(
J+1)=U:K=1
130 NEXT J
140 IF K<>0 THEN 100
200 FOR J=0 TO 19:POSITION 9,J:? Z(J);:NEXT J
```

30: Die Zufallszahlen sind echt.

50: Der Bildschirm wird mit CHR$(125) gelöscht.

60: Der Cursor wird mit POSITION an die gewünschte Bildschirmposition gestellt. Und ein Ausrichten der Zahlen ist nicht möglich, weil Atari keinen PRINT USING-Befehl hat.

120: Atari-BASIC kennt keinen SWAP-Befehl. Was nun? Wenn wir den Inhalt der einen Variablen in die andere schreiben, Z(J)=Z(J+1), dann haben beide den Wert von Z(J+1), und der Inhalt von Z(J) ist verloren.
Also müssen wir den Inhalt von Z(J) zuerst in einer *Hilfsvariablen* zwischenlagern. U bekommt den Wert von Z(J). Dann bekommt Z(J) den Wert von Z(J+1). Und zuletzt wird der alte Werte von Z(J), der jetzt nur noch in U enthalten ist, Z(J+1) zugeordnet: ein Ringtausch also.

Auch Commodore meint, ohne SWAP und PRINT USING auszukommen. Zusätzliche Unannehmlichkeiten ziehen auf, weil es

keinen Befehl gibt, um den Cursor auf eine bestimmte Bildschirmposition zu stellen. Commodore beschreibt den Bildschirm übrigens mit 25 *Zeilen* statt mit 24.

```
0 REM SORTZAHL.COM
10 DIM Z(19),A(19):R=RND(-TI)
20 FOR J=0 TO 19
30 Z(J)=INT(RND(1)*1000)
40 A(J)=Z(J)
50 NEXT J
100 K=0
110 FOR J=0 TO 18
120 IF Z(J)>Z(J+1) THEN U=Z(J):Z(J)=Z(J+1):Z(
J+1)=U:K=1
130 NEXT J
140 IF K<>0 THEN 100
200 PRINTCHR$(147):FOR J=0 TO 19:PRINT A(J);T
AB(10);Z(J):NEXT J
```

40: Um Komplikationen aus dem Wege zu gehen, erfassen wir jede Zufallszahl gleich zweimal; einmal in Z(J), die dann später sortiert wird, und einmal in A(J), die ungeordnet bleibt.

200: Beide Zahlenreihen werden erst nach dem Sortieren ausgegeben.

Eine bestimmte Bildschirmposition kann bei Commodore nur beschrieben werden, indem ein *Code* für das gewünschte Zeichen *direkt im Bildschirmspeicher* abgelegt wird. In späteren Programmen werden wir nicht darum herumkommen, uns damit zu befassen. An dieser Stelle mag der gezeigte Kompromiß hinreichen.

Das Sortieren ist eine vielbenötigte Routine. So wie hier numerische Werte sortiert werden, können auch Stringvariablen, also z. B. Familiennamen, alphabetisch geordnet werden. Natürlich muß dann mit einer *String-Feldvariablen* gearbeitet werden.

Die vorgestellte Sortiermethode ist nicht die einzige. Sie ist auch nicht die beste, sprich schnellste. Aber sie ist die simpelste. Am Ende des Buches, beim Programm CODEKNCK, werden wir auf diese kleine Übung hier mit Freude zurückgreifen.

3 Farbenspiele

Wo bislang trockene Zahlen und Buchstaben über die Mattscheibe flimmerten, soll es jetzt bunt zugehen. Allerdings geht es dann auch »bunt« in den BASIC-Dialekten zu.

Um die Daten des Rechners so zu verarbeiten, daß der Bildschirm sie empfangen und in bunte Punkte umsetzen kann, bedarf es einer hochkomplizierten Technik. Jeder Computer verfügt über einen Video-Prozessor, der die Verbindung zum Fernsehgerät beziehungsweise Monitor besorgt.

Es gibt eine Vielzahl solcher Prozessoren, und die verschiedenen Heimcomputer sind mit wechselnden Bausteinen bestückt. Manche Video-Chips leisten mehr als andere, und entsprechend bieten sich für die Konstrukteure andere Möglichkeiten für die Bildschirmdarstellung, die wiederum zum Bedarf an bestimmten BASIC-Befehlen führt.

Das Ergebnis dieser Vielfalt zeigt sich darin, daß die *fundamentalsten* Unterschiede der diversen BASIC-Dialekte im Bereich der *Grafik* zu finden sind.

Trotzdem gibt es eine Reihe von Grundlagen, die für alle Systeme bindend sind. Die Bildröhre arbeitet *zeilenweise*. Ein *Kathodenstrahl* tastet die Bildfläche ab. Er beginnt in der Ecke oben links und bewegt sich über die Bildschirmzeile nach rechts. Am rechten Rand angekommen, wird er kurz abgeschaltet (HBLANK = horizontal blank), springt an den linken Rand der nächst tieferen Zeile, wird wieder angeschaltet und wandert wieder nach rechts. Ist der Strahl in der Ecke unten rechts angelangt, wird er ausgeschaltet (VBLANK = vertical blank), um zurück in die obere linke Ecke zu springen. Dieser gesamte Ablauf wird fünfzigmal in einer Sekunde

bearbeitet. Wenn diese Darstellung auch für einen Techniker haarsträubend vereinfacht sein mag, so reicht sie doch für unser Verständnis von der Bildtechnik aus.

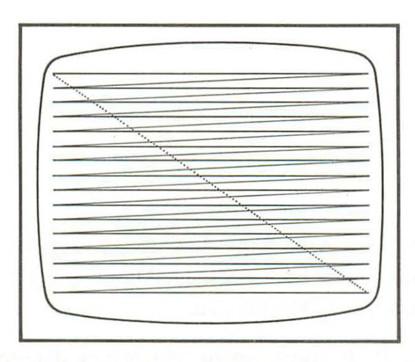


Abb. 3: Der Kathodenstrahl tastet die Bildröhre Zeile um Zeile ab

Der Rechner muß sich der Arbeitsweise der Bildröhre anpassen. Aber den unvorstellbaren Informationsbedarf eines normalen Fernsehbildes kann heute noch nicht einmal ein größerer Bürocomputer bedienen. Das Fernsehbild besteht nach der bei uns gültigen CCIR-Norm aus 625 Zeilen mit insgesamt einer halben Million Bildpunkte. Selbst wenn jeder dieser Punkte nur einfach leuchten oder nicht-leuchten sollte, was einem Bit Information entspräche, würde ein Bildschirminhalt runde 122 kByte verschlingen, und das fünfzigmal pro Sekunde.

Heimcomputer, wie sie heute mit 64 kByte angeboten werden, lassen dem Programmierer rund 32 kByte frei. Der Rest wird vom System selbst belegt. Man würde also theoretisch vier Heimcomputer benötigen, um ein einfaches Schwarzweiß-Bild ohne Grautöne zu speichern.

Heimcomputer bearbeiten deshalb nur einen Teil der Bildröhre. Meist sind es 192 Zeilen, bei Commodore z. B. 200. Und auf jeder Zeile werden nur 320, bei MSX gar nur 256 Bildpunkte *(Pixel)* gesetzt.

Eine solche Fläche von 320 mal 192 Punkten zu verwalten, beansprucht Heimcomputer schon so sehr, daß Farbwahl meist nicht mehr möglich ist; die Punkte sind entweder an oder aus, leuchten oder leuchten nicht.

Um mit weniger Speicherplatz auszukommen, werden *größere Einheiten* als Pixel verarbeitet. So kann ein Pixel z. B. acht Bildpunkte breit und acht Fernsehzeilen hoch sein. Die Auflösung beträgt dann 40 mal 24 (oder bei Commodore 25) Bildpunkte; das sind 960 beziehungsweise 1000 Pixel, eine überschaubare Menge.

Neben der Information, ob ein Pixel gesetzt oder nicht gesetzt ist, muß der Rechner noch speichern, *welche Farbe* das Pixel annehmen soll, wenn mehr als zwei Farben (EIN oder AUS) möglich sind. Dadurch vervielfacht sich die Informationsmenge entsprechend.

Soll Text auf dem Bildschirm gezeigt werden, so verwaltet der Rechner 960 (beziehungsweise 1000) Schreibstellen und merkt sich für jede dieser Bildschirmpositionen eine Zahl, die dem Zeichen entspricht, das an dieser Stelle dargestellt werden soll. Leider arbeiten die Rechner intern selten mit den ASCII-Codes; sondern jeder Typ hat einen eigenen internen Code, der jedem verfügbaren Zeichen seines Zeichensatzes einen Zahlenwert, meist von 0 bis 255, zuordnet.

Jedes Zeichen, das der Rechner auf dem Bildschirm darstellen kann, besteht aus einer *Punktematrix* von acht mal acht Bildpunkten. Natürlich sehen diese Punktemuster und damit die Zeichen bei jedem Rechnertyp anders aus.

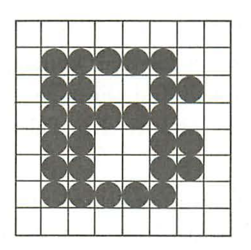


Abb. 4: Ein Buchstabe, dargestellt in einer Matrix aus acht mal acht Bildpunkten

Diese Muster von Bildpunkten sind in Zahlenwerten erfaßt und im Rechner abgespeichert. Mit dem internen Code werden die Daten für das Punktemuster gefunden und wird so das Zeichen auf dem Bildschirm dargestellt.

Im Rechner gespeicherte Grafikdaten werden vom Video-Prozessor gelesen, in Fernsehsignale umgewandelt und synchron mit dem Arbeitstakt der Bildröhre gesendet, also fünfzigmal in der Sekunde.

Beim *Commodore 64* ist die Aufgabe des Bildschirmspeichers besonders simpel gelöst. Es gibt einen Speicherbereich, in dem für jede der tausend Schreibstellen des Bildschirms der Zeichencode gespeichert wird. Und in einem zweiten Speicherbereich wird für jede dieser Schreibstellen ein Farbwert abgelegt. Beide Speicherbereiche sind voneinander unabhängig; eine Bildschirmdarstellung ermöglichen sie aber nur im Zusammenwirken. Denn wenn ein Zeichen bestimmt wird, aber keine Farbe da ist, dann ist nichts zu sehen und umgekehrt.

Bei dieser Speicherorganisation ist hochauflösende Grafik, also gezieltes Ansprechen eines beliebigen der 320 mal 200 Pixel, nur mit viel Mühe und bester Kenntnis des Commodore-Innenlebens möglich.

Atari geht bei der Bildgestaltung ganz andere Wege. Sechzehn verschiedene Grafikmodi stehen zur Auswahl. Sie unterscheiden sich darin, ob Text, also Zeichen aus dem Zeichensatz, oder Grafik, also einfache Bildpunkte, dargestellt werden, wie viele verschiedene Farben möglich sind und, bei Grafik, wie viele Pixel die Bildfläche enthält, also die Auflösung. Ein Bild kann aber immer nur aus Pixel bestimmter Größe und verschiedener Farben oder aus Schriftzeichen mit stark eingeschränkten Farbmöglichkeiten bestehen.

In den meisten Grafikmodi des Atari können nur vier verschiedene Farben gleichzeitig auf dem Bildschirm gezeigt werden. Aber diese vier Farben können aus einer reichen Palette von 128 Farbtönen frei gewählt werden.

Das *MSX-System* hat zwei verschiedene Textbetriebsarten und zwei Grafikmodi. Bei Textbetrieb ist keine Farbwahl möglich. In dem einen Grafikgang sind die Pixel vier mal vier Bildpunkte groß, und für jedes einzelne kann einer von sechzehn Farbwerten bestimmt werden. In der hochauflösenden Grafik werden 256 mal 192 Pixel angesprochen; aber die Farbmöglichkeiten sind stark eingeschränkt.

Charakterisierend kann man sagen, daß Commodore überlegen ist, wenn mit einer Auflösung von 40 mal 25 gearbeitet wird, weil alle

Zeichen und Farben in einem Bild frei gemischt werden können. Atari hat seine Stärke in höher aufgelöster Grafik, die auch für einen BASIC-Programmierer und Anfänger leicht zu verstehen ist. MSX schneidet bei der Bildgestaltung am schlechtesten ab.

3.1 Wechselrahmen

Als kleine Fingerübung wollen wir farbige quadratische Rahmen auf den Bildschirm zaubern. Dabei soll der Rechner die Größe und Farbe der Rahmen per Zufall bestimmen und im endlosen Wechsel neu schreiben, so daß ein sich ständig veränderndes Bild entsteht.

In der Atari-Version sieht das Programm so aus:

```
0 REM QUADRATE.ATA
10 GRAPHICS 19
20 POKE 712,0:POKE 708,52:POKE 709,68:POKE 71
0,230
30 C=INT(RND(0)*3)+1
40 X=INT(RND(0)*12)
50 COLOR C
60 PLOT 19-X,11-X
70 DRAWTO 19-X,11+X
80 DRAWTO 19+X,11+X
90 DRAWTO 19+X,11-X
100 DRAWTO 19-X,11-X
110 FOR Z=0 TO 300:NEXT Z
120 GOTO 20
```

10: Hier wird eine Grafikauflösung von 40 mal 24 Pixel gewählt. In dieser Betriebsart können vier Farbtöne gleichzeitig auf dem Bildschirm gezeigt werden.

20: In die Farbregister 708, 709, 710 und 712 werden die gewünschten Farbwerte geschrieben. Die gleiche Wirkung erreicht man auch mit dem **SETCOLOR**-Befehl.

30: Die vier bestimmten Farbwerte werden mit dem Befehl **COLOR** 0 bis 3 angesprochen. In dieser Zeile wird ein zufälliger Wert von 1 bis 3 für COLOR gefunden. COLOR 0 ist die Hintergrundfarbe, in diesem Fall Schwarz, die wir zur Darstellung nicht verwenden wollen.

40: Auch die Ausmaße des Rahmens werden vom Zufall bestimmt. Der Bildschirm sieht bei dieser Computergrafik so aus:

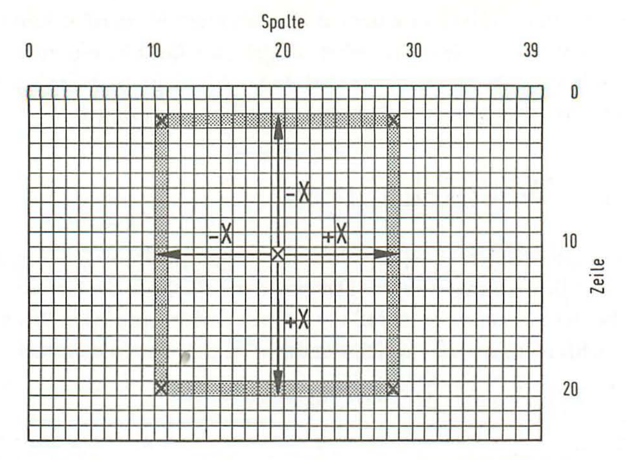


Abb. 5: Bildschirmgestaltung bei Programm QUADRATE.ATA

In der linken oberen Ecke des Bildschirms liegt der Punkt 0,0. Der Mittelpunkt unserer Grafik soll bei 19,11 liegen. Der Rahmen kann dann maximal 23 mal 23 Pixel groß werden. Die Abweichung vom Mittelpunkt beträgt +/− 11, und da der Rahmen quadratisch sein soll, muß die Abweichung in X-Richtung (Spalten) und Y-Richtung (Zeilen) jeweils gleich sein. Der Wert für die Abweichung wird in dieser Zeile zufällig festgesetzt.

50: Der zufällig gewählte Farbwert wird aktiviert.

60: Mit **PLOT** wird ein Grafikpunkt an eine bestimmte Stelle des Bildschirms gesetzt.

70 bis 90: **DRAWTO** setzt voraus, daß zuvor ein Punkt bearbeitet (PLOT oder DRAWTO) wurde. DRAWTO zieht dann eine Linie von diesem Punkt zu dem neu bestimmten Punkt.

110: In dieser FOR...NEXT-Schleife geschieht nichts weiter. Sie ist *reine Beschäftigungstherapie* für den Rechner, der sonst in einer Geschwindigkeit Rahmen auf den Monitor pinseln würde, daß es nicht mit anzusehen wäre. Löschen Sie diese Zeile, und überzeugen Sie sich selbst!

120: Ein farbiger Rahmen ist gezeichnet; das Programm springt zurück, um einen neuen Rahmen zu berechnen. So kann das Programm kein Ende finden. Wenn Sie nicht den Stecker herausziehen,

BREAK oder **RESET** drücken, so zaubert der Rechner bunte Bilder auf Ihren Fernseher, bis seine Schaltkreise durchbrennen. Eigentlich wäre es korrekter, wenn der Rücksprung hier nach Zeile 30 erfolgen würde, denn in Zeile 20 werden die Farbwerte definiert; solches muß aber nicht immer wieder neu gemacht werden. Es stört aber auch nicht, wenn es erfolgt. Eine unnötige, wenn auch unerhebliche Belastung für den Rechner; ein Beispiel aber auch, daß BASIC kleinere Unkorrektheiten unter günstigen Bedingungen durchaus verkraftet. Und das macht diese Programmiersprache so sympathisch.

Bei MSX wählen wir mit **SCREEN 3** die Betriebsart mit einer Auflösung von 80 mal 48 Bildpunkten. Die Besonderheit ist hier, daß der Bildschirm trotzdem in Koordinaten von 0 bis 255 in X-Richtung und 0 bis 191 in Y-Richtung aufgeteilt ist. Jeder Grafikpunkt wird als Block aus vier mal vier Koordinatenpunkten bearbeitet und kann über jede einzelne dieser sechzehn Koordinaten angesprochen werden:

```
0  REM QUADRATE.MSX
10 SCREEN 3:X=RND(-TIME)
20 C=INT(RND(1)*16)
30 K=INT(RND(1)*92)
40 LINE (127-K,95-K)-(128+K,96+K),C,B
50 LINE (123-K,91-K)-(132+K,100+K),C,B
60 FOR W=0 TO 300:NEXT W
70 GOTO 20
```

20: Es stehen sechzehn Farben zur Auswahl. Diemal lassen wir für die Rahmen auch die Hintergrundfarbe zu.

30: Abweichungen vom Zentrum bei 127,91 werden zufällig von 0 bis 91 festgelegt.

40: Der **LINE**-Befehl zieht eine Linie zwischen den beiden bestimmten Punkten. Der Wert C bestimmt die Farbe für die Linie. Das angehängte B (box = Kasten) bewirkt, daß nicht eine direkte Linie gezogen, sondern ein Viereck gezeichnet wird, das die beiden bestimmten Punkte als gegenüberliegende Ecken hat. Ersetzen Sie B durch BF, und sehen Sie, was geschieht!

50: Da die Auflösung hier doppelt so fein ist wie bei Atari und Commodore, ziehen wir noch einen zweiten Kasten gleich daneben, um ein ähnliches Aussehen zu erreichen.

60: Nach einer kurzen Augenpause ...

70: führt GOTO zurück für einen neuen Durchlauf.

Um das gleiche kleine Programm für Commodore zu schreiben, müssen wir uns nun ein wenig mit dem Innenleben des Rechners beschäftigen. Er besteht aus einer sehr großen Zahl von Speicherzellen, die jede acht Bit, das ist ein Byte, Information fassen kann, also Werte zwischen 0 und 255 (dezimal).

Um Ordnung in die Sache zu bringen, sind alle Speicherplätze fortlaufend numeriert. Beim Commodore beginnt der Bildschirmspeicher für die Zeichen mit der Speicherzelle 1024. Dort liegt der Zeichencode für das Zeichen, das in der oberen linken Ecke des Bildschirms dargestellt wird. In der folgenden Speicherzelle liegt der Code für das nächste Zeichen auf der gleichen Zeile, und so weiter, bis das vierzigste Zeichen erreicht ist, das in der Ecke oben rechts des Bildschirms liegt: Speicherplatz 1063, nämlich 1024+39. Mit der Speicherzelle 1024+40 befinden wir uns wieder am linken Bildschirmrand, aber in der nächsttieferen Zeile.

Ein Bildschirmpunkt, der in der X-ten Spalte liegt, gehört zu der Speicherzelle 1024+X; ein Punkt in der Y-ten Zeile wird in der Speicherzelle 1024+(Y*40) verwahrt.

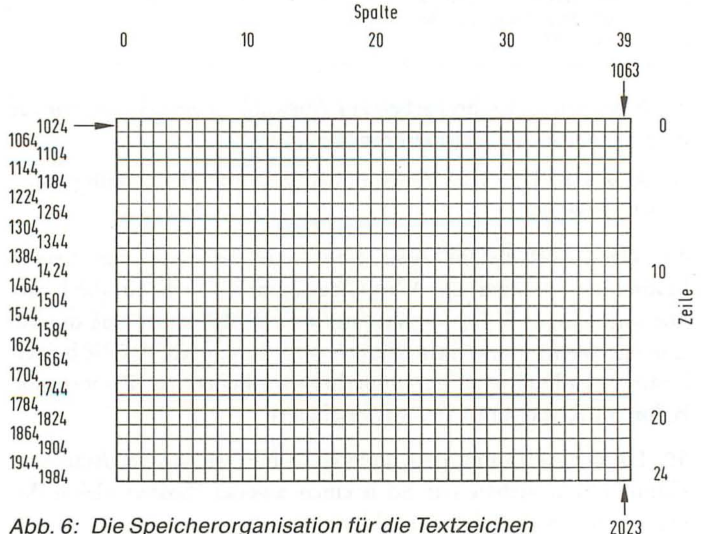


Abb. 6: Die Speicherorganisation für die Textzeichen

Jetzt können wir aus X- und Y-Koordinaten einer Bildschirmposition ohne große Mühe die zugehörige Speicherzelle errechnen. Wir müssen nur daran denken, Zeilen und Spalten mit Null beginnend zu zählen. Wenn der Punkt die Koordination 7,12 hat, dann gehört die Speicherzelle 1024+7+(12*40)=1511 dazu. Oder allgemein formuliert:

Bildschirmposition X,Y → Speicherzelle 1024+X+Y*40

Eine Klammer ist nicht nötig, weil die Multiplikation Vorrang hat und zuerst ausgeführt wird.

Der *Farbspeicher* ist in gleicher Weise aufgebaut, nur daß in den einzelnen Zellen jeweils der Farbcode (0 bis 15) für den entsprechenden Bildschirmpunkt abgelegt ist. Die Basisadresse des Farbspeichers ist 55296.

Andere Computer arbeiten übrigens nach dem gleichen Prinzip. Nur muß man sich als Hobbyprogrammierer nicht damit befassen, weil die verfügbaren BASIC-Befehle diese Arbeit übernehmen. Wenn also bei Atari ein **PLOT X,Y** ergeht, dann kalkuliert das System selbst, in welcher Speicherzelle die Bildinformation abgelegt werden muß. Weil das »Spar-BASIC« des Commodore keine Grafikbefehle enthält, kann nur direkt in den Bildschirmspeicher geschrieben werden, was man bei MSX oder Atari genauso machen könnte, wenn man wollte.

```
0 REM QUADRATE.COM
10 T=1024:F=55296:R=RND(-TI)
20 PRINTCHR$(147):POKE 53280,0:POKE 53281,0
30 FOR I=7 TO 31:FOR J=0 TO 24:POKE T+J*40+I,
81:NEXT J:NEXT I:REM 160
40 C=INT(RND(1)*16)
50 K=INT(RND(1)*13)
60 FOR X=19-K TO 19+K:POKE F+(12-K)*40+X,C:PO
KE F+(12+K)*40+X,C:NEXT X
70 FOR Y=12-K TO 12+K:POKE F+19-K+Y*40,C:POKE
 F+19+K+Y*40,C:NEXT Y
80 FOR W=0 TO 300:NEXT W
90 GOTO 40
```

10: Die Basisadressen der beiden Bildschirmspeicherbereiche werden in den Variablen T und F erfaßt, da diese Werte im folgenden mehrfach gebraucht werden.

20: Mit dem Befehl **POKE** wird in die bestimmte Speicheradresse ein bestimmter Wert geschrieben. Beim Commodore hält die Adresse 53280 den Farbwert für den Rand und 53281 den des Hintergrundes.

30: Damit etwas auf dem Bildschirm sichtbar werden kann, muß an der angesprochenen Stelle ein Zeichen stehen. Wir setzen ein Grafikzeichen, einen dicken Punkt, mit dem Zeichencode 81. Die *negative Leerstelle*, also ein Zeichen, das die gesamte Schreibstelle ausfüllt, hat den Zeichencode 160. Experimentieren Sie hier mit verschiedenen Werten für den Zeichencode. Da auf dem Bildschirm ein Quadrat erscheinen soll, wird nur ein Teil des Bildschirmspeichers beschrieben, nämlich von Spalte 7 bis 31 und von Zeile 0 bis 24. Zu sehen ist jetzt aber noch nichts, da noch keine Farbwerte bestimmt sind.

40 und 50: Farbwert und Abweichung werden genauso ermittelt, wie es für Atari erklärt wurde, nur daß mehr Farben zur Auswahl stehen und der Bildschirm eine Zeile höher ist. Der Mittelpunkt liegt jetzt bei 19,12, und die Abweichungen können +/− 12 betragen.

60 und 70: Die Eckpunkte des Rahmens finden wir wie bei Atari erklärt. Nur müssen wir *jede einzelne* Bildschirmposition mit dem Farbwert beschreiben. Eine FOR...NEXT-Schleife übernimmt diese Arbeit. Die erste Schleife bearbeitet den oberen und unteren Schenkel des Rahmens, die zweite die beiden seitlichen.

3.2 Computerkaleidoskop

Zugegeben, die farbigen Rahmen zu betrachten, wird man schnell müde. Deswegen wollen wir jetzt unsere grauen Zellen etwas mehr anstrengen und das Bildergebnis anspruchsvoller ordnen.

Das Kaleidoskop-Programm ermittelt einen zufälligen Punkt und *spiegelt* diesen an den vier orthogonalen und diagonalen Symmetrieachsen. Das hört sich komplizierter an, als es tatsächlich ist.

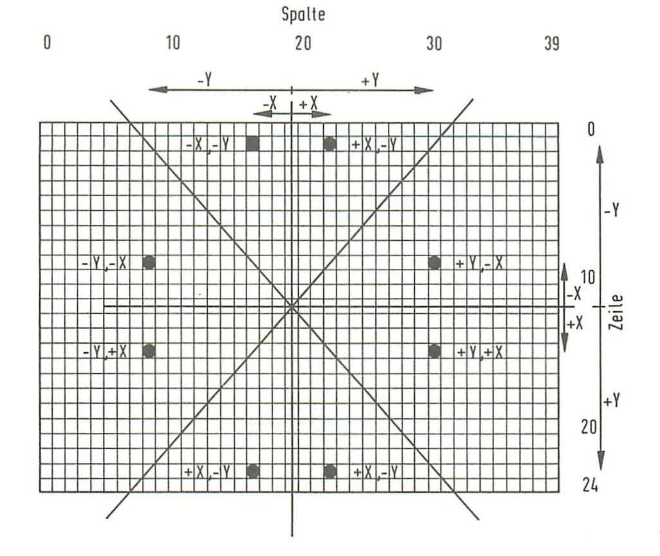


Abb. 7: Nur der quadratisch gezeichnete Punkt wird zufällig ermittelt. Die übrigen sieben Punkte werden durch die Symmetriegesetze bestimmt

Sehen Sie sich in der Abbildung 7 den Entwurf für die Bildschirmgrafik (Commodore) an. Der *Mittelpunkt* ist wieder bei 19,12 gewählt.

Das Programm ermittelt zwei zufällige Werte für die Abweichung vom Mittelpunkt, die hier X und Y genannt wurden. Der in der Abbildung quadratisch gezeichnete Punkt wird dann durch 19−X, 12−Y gefunden.

Einen zweiten, an der *senkrechten* Achse gespiegelten Punkt erhalten wir ganz einfach dadurch, daß wir X addieren, statt zu subtrahieren: 19+X, 12−Y.

Zwei weitere Punkte findet die Spiegelung an der *waagerechten* Achse, also wenn Y addiert statt subtrahiert wird: 19−X, 12+Y und 10+Y,12+Y.

Die Spiegelung an einer *Diagonalen* ist genauso einfach zu berechnen; X- und Y-Werte müssen lediglich vertauscht werden, also 19−Y,12−X usw.

Das Commodore-Programm für das Kaleidoskop sieht ähnlich aus wie QUADRATE.COM:

```
0 REM KALEIDOS.COM
10 T=1024:F=55296:R=RND(-TI):C=1
20 PRINT CHR$(147):POKE 53280,0:POKE 53281,0
30 FOR J=0 TO 24:FOR I=7 TO 31:POKE T+J*40+I,
160:NEXT I:NEXT J:REM 102
40 FOR I=7 TO 31:FOR J=0 TO 24:POKE F+J*40+I,
11:NEXT J:NEXT I
70 C=INT(RND(1)*15)+1
80 X=INT(RND(1)*13)
90 Y=INT(RND(1)*13)
100 POKE F+(12-X)*40+19-Y,C
110 POKE F+(12-X)*40+19+Y,C
120 POKE F+(12+X)*40+19+Y,C
130 POKE F+(12+X)*40+19-Y,C
140 POKE F+(12-Y)*40+19-X,C
150 POKE F+(12-Y)*40+19+X,C
160 POKE F+(12+Y)*40+19+X,C
170 POKE F+(12+Y)*40+19-X,C
180 GOTO 70
```

30: Versuchen Sie statt der vollen Leerstelle einmal den Zeichencode 102.

40: Hier wird für alle vom Programm bearbeiteten Schreibstellen ein Farbwert in den Farbspeicher geschrieben. Dadurch wird die gesamte Fläche auf dem Bildschirm sichtbar.

70 bis 90: Zufällige Werte für Farbe, X- und Y-Abweichung werden ermittelt.

100 bis 170: Wie die acht symmetrisch angeordneten Bildschirmpositionen gefunden werden, haben wir uns oben deutlich gemacht. Hier werden die entsprechenden Speicherzellen berechnet und wird der Farbwert C hineingeschrieben (POKE).

180: Auch dieses Programm läuft durch den Rücksprung nach 70 endlos.

In der Atari-Fassung werden die acht Grafikpunkte mit PLOT auf den Bildschirm gebracht:

```
0 REM KALEIDOS.ATA
10 GRAPHICS 19
20 POKE 712,2:POKE 708,6:POKE 709,8:POKE 710,
12
30 FOR J=0 TO 22:FOR I=8 TO 30:C=INT(RND(0)*4
```

```
):COLOR C:PLOT I,J:NEXT I:NEXT J
40 FOR Z=0 TO 300:NEXT Z
50 POKE 712,0:POKE 708,146:POKE 709,68:POKE 7
10,230
60 C=INT(RND(0)*3)+1
70 X=INT(RND(0)*12)
80 Y=INT(RND(0)*12)
90 COLOR C
100 PLOT 19-X,11-Y:PLOT 19+X,11+Y
110 PLOT 19+X,11-Y:PLOT 19-X,11+Y
120 PLOT 19-Y,11-X:PLOT 19+Y,11+X
130 PLOT 19+Y,11-X:PLOT 19-Y,11+X
140 FOR Z=0 TO 100:NEXT Z
150 GOTO 60
```

30: Eine kleine zusätzliche Spielerei; die gesamte Bildfläche wird vorab mit zufälligen Farbwerten gefüllt. Diese Zeile können Sie ersatzlos streichen.

90: Die zufällig ermittelte Farbe wird aktiviert.

100 bis 130: Dann werden die acht symmetrisch verteilten Punkte auf dem Bildschirm gesetzt.

140: Da das Programm sehr schnell arbeitet, ist eine Warteschleife sinnvoll.

150: Rücksprung.

Bei MSX wird es nur deshalb etwas umständlich, weil die Farbpunkte die gleiche Größe bekommen sollen, wie in den beiden anderen Versionen. Da die Grafikpunkte in SCREEN 3 aber nur ein Viertel so groß sind, müssen jeweils vier Punkte gesetzt werden:

```
0  REM KALEIDOS.MSX
10 SCREEN 3:X=RND(-TIME)
20 COLOR 15,1,1:CLS
30 FOR I=32 TO 223 STEP 8:FOR J=0 TO 191 STEP
 8:C=INT(RND(1)*2)+14:PSET (I,J),C:PSET(I+4,J
),C:PSET(I+4,J+4),C:PSET(I,J+4),C:NEXT J: NEX
T I
70 C=INT(RND(1)*10)+1
80 X=(INT(RND(1)*12)+1)*8-1
90 Y=(INT(RND(1)*12)+1)*8-1
100 PSET(127-X,95-Y),C:PSET(131-X,95-Y),C:PSE
T(131-X,99-Y),C:PSET(127-X,99-Y),C
```

```
110 PSET(127-Y,95-X),C:PSET(131-Y,95-X),C:PSE
T(131-Y,99-X),C:PSET(127-Y,99-X),C
120 PSET(128+X,96+Y),C:PSET(124+X,96+Y),C:PSE
T(124+X,92+Y),C:PSET(128+X,92+Y),C
130 PSET(128+Y,96+X),C:PSET(124+Y,96+X),C:PSE
T(124+Y,92+X),C:PSET(128+Y,92+X),C
140 PSET(128+X,95-Y),C:PSET(124+X,95-Y),C:PSE
T(124+X,99-Y),C:PSET(128+X,99-Y),C
150 PSET(128+Y,95-X),C:PSET(124+Y,95-X),C:PSE
T(124+Y,99-X),C:PSET(128+Y,99-X),C
160 PSET(127-X,96+Y),C:PSET(131-X,96+Y),C:PSE
T(131-X,92+Y),C:PSET(127-X,92+Y),C
170 PSET(127-Y,96+X),C:PSET(131-Y,96+X),C:PSE
T(131-Y,92+X),C:PSET(127-Y,92+X),C
180 GOTO 70
```

30: Auch hier werden vorab zufällige Farbwerte gezeichnet.

80 und 90: Die Abweichung kann bis zu elf Kaleidoskop-Punkte betragen, das sind 22 Grafikpunkte von SCREEN 3 oder 88 Koordinatenwerte. Der zufällig gefundene Abweichungswert wird deshalb mit acht multipliziert.

100 bis 170: An jede gefundene Stelle wird mit **PSET** ein Grafikpunkt auf dem Bildschirm gesetzt und rechts daneben und darunter weitere drei Punkte, um auf die gewünschte Größe zu kommen.

180: Endlosschleife durch Rücksprung.

3.3 Malwunder

Im nächsten Schritt wollen wir die grafischen Veränderungen auf dem Bildschirm selbst in die Hand nehmen. Die Verteilung der Bildpunkte und ihrer Farben soll nicht mehr dem Zufall überlassen bleiben, sondern der Benutzer soll nach *seinen* Wünschen auf dem Bildschirm malen können.

Dazu müssen wir eine Möglichkeit schaffen, in das laufende Programm Befehle einzugeben. Für diesen Fall bietet sich die Eingabe über *Steuerknüppel* an.

Es gibt kaum einen Heimcomputer, der keine Anschlußbuchsen für *Joysticks* (Steuerknüppel) aufweist. Die Vielfalt der Systeme hat sich in letzter Zeit reduziert; und der vom Videospielespezialisten

Atari eingeführte neunpolige Stecker findet heute allgemein Verwendung.

Der Steuerknüppel ist durchaus vergleichbar mit der Gangschaltung beim Auto. Üblicherweise erkennt der Joystick neben der Ruhestellung acht weitere Positionen. Je nachdem, in welche Lage er gedrückt wird, schaltet die Mechanik des Steuerknüppels bestimmte Leitungen, was vom Rechner erkannt und in Zahlenwerte umgesetzt wird.

Bei Atari findet die Funktion **STICK(n)** je nach Zustand des Steuerknüppels einen anderen Zahlenwert. Bei den neueren Modellen können *zwei* Joysticks angeschlossen und gelesen werden, STICK(0) und STICK(1). Bei den älteren Modellen waren es *vier*.

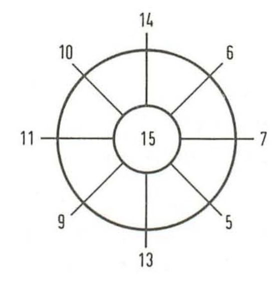


Abb. 8: Die Zuordnung von Zahlenwerten zu Richtungen beim Atari-Steuerknüppel

Die Abbildung zeigt, welchen Zahlenwert die STICK-Funktion je nach Stellung des Steuerknüppels findet. Für den Nichttechniker sieht die Zuordnung der Zahlen zu den Richtungen ganz willkürlich aus. Schriebe man die Werte in binärer Form, würde sich die dahinterstehende Logik jedoch deutlich zeigen. Das muß uns an dieser Stelle aber nicht weiter beschäftigen.

Der Joystick hat auch noch einen *Feuerknopf* (trigger), der mit der Funktion **STRIG(n)** gelesen wird. Ist der Feuerknopf gedrückt, findet STRIG(n) den Wert 0, sonst den Wert 1.

Im folgenden Programm können Sie durch Bewegen des Steuerknüppels eine Linie ziehen, die, wie für das Kaleidoskop erklärt,

vierfach gespiegelt wird; das Drücken des Feuerknopfes ändert die Farbe:

```
0 REM KALEMOVE.ATA
10 GRAPHICS 19:POKE 712,0:POKE 708,68:POKE 70
9,96:POKE 710,234:C=1
20 IF STRIG(0)=0 THEN C=C+1:IF C>3 THEN C=0:F
OR W=0 TO 100:NEXT W
30 S=STICK(0)
40 IF S=7 THEN X=X+1:IF X>11 THEN X=11
50 IF S=11 THEN X=X-1:IF X<0 THEN X=0
60 IF S=13 THEN Y=Y+1:IF Y>11 THEN Y=11
70 IF S=14 THEN Y=Y-1:IF Y<0 THEN Y=0
80 COLOR C
100 PLOT 19-X,11-Y:PLOT 19+X,11+Y
110 PLOT 19+X,11-Y:PLOT 19-X,11+Y
120 PLOT 19-Y,11-X:PLOT 19+Y,11+X
130 PLOT 19+Y,11-X:PLOT 19-Y,11+X
140 GOTO 20
```

10: Der Grafikmodus wird ausgewählt; in den Farbregistern werden die gewünschten Farbwerte abgelegt; die Variable C wird auf 1 gesetzt, die später die Farbe aktivieren soll.

20: Hier wird der Feuerknopf abgefragt. Wenn der Feuerknopf gedrückt ist (=0), dann wird C um 1 erhöht. Da Farben nur von COLOR 0 bis 3 aufgerufen werden können, müssen wir verhindern, daß C größer als 3 wird. Wenn C durch Betätigung des Feuerknopfes einen größeren Wert annehmen würde, dann bekäme C wieder den Wert 0.

30: Da das Wort STICK(0) aus immerhin acht Zeichen besteht, ordnen wir den Wert von STICK(0) der Variablen S zu, die sich im folgenden einfacher schreibt und weniger Platz im Speicher belegt.

40: Wenn der Steuerknüppel nach rechts gedrückt ist, hat S den Wert 7. Wenn der Grafikpunkt nach rechts wandern soll, dann müssen wir den Koordinatenwert für die X-Richtung um 1 erhöhen: X=X+1. Da die maximale Abweichung vom Zentrum, wie bei den vorherigen Programmen, 11 beträgt, müssen wir noch dafür sorgen, daß X nicht größer als 11 werden kann. Natürlich könnte auch hier etwas anderes bestimmt werden – z. B. daß X wieder auf 0 gesetzt wird, wenn es größer als 11 werden will, daß das Spiel endet, oder daß etwas Beliebiges geschieht.

50 bis 70 fragen auf die gleiche Weise die übrigen drei Richtungen ab.

80: Dann wird die aktuelle Farbe durch den Wert der Variablen C aktiviert.

100 bis 130: Und schließlich werden die acht symmetrischen Punkte gesetzt. Da jeder neue Punkt immer neben dem vorherigen liegt, entsteht eine durchgehende Linie. Der Benutzer bewegt den Joystick und zeichnet, knopfdrückend die Farbe wechselnd, gleichzeitig acht symmetrische Linien auf den Bildschirm.

140: Der Rücksprung erfolgt zur Abfrage des Feuerknopfes in Zeile 30.

Jeder Atari-kompatible Joystick kann an einen MSX-Computer angeschlossen werden. Er wird mit dem gleichen Befehl gelesen; STICK(1) und STICK(2). STICK(0), eine Besonderheit, liest die vier Cursortasten des Computers, so daß ein MSX-Besitzer auch *ohne* Steuerknüppel auskommen kann.

Die Zahlenwerte sind mit Rücksicht auf den BASIC-Programmierer in aufsteigender Reihenfolge im Uhrzeigersinn angeordnet.

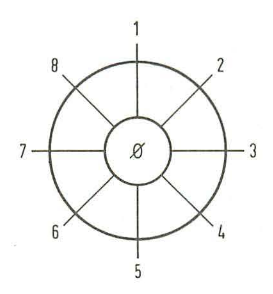


Abb. 9: Die Zuordnung von Zahlenwerten zu Richtungen bei MSX

Der Feuerknopf wird auch bei MSX mit STRIG(n) gelesen und findet den Wert −1, wenn gedrückt, und 0, wenn nicht gedrückt. Analog zu den Cursortasten liest STRIG(0) die Leertaste.

```
0   REM KALEMOVE.MSX
10 SCREEN 3:C=8
20 IF STRIG(0)=-1 THEN C=C+1:IF C>15 THEN C=1
30 S=STICK(0)
40 IF S=5 THEN X=X-1:IF X<1 THEN X=1
50 IF S=1 THEN X=X+1:IF X>12 THEN X=12
60 IF S=3 THEN Y=Y-1:IF Y<1 THEN Y=1
70 IF S=7 THEN Y=Y+1:IF Y>12 THEN Y=12
100 PSET(127-X*8,95-Y*8),C:PSET(131-X*8,95-Y*
8),C:PSET(131-X*8,99-Y*8),C:PSET(127-X*8,99-Y
*8),C
110 PSET(127-Y*8,95-X*8),C:PSET(131-Y*8,95-X*
8),C:PSET(131-Y*8,99-X*8),C:PSET(127-Y*8,99-X
*8),C
120 PSET(128+X*8,96+Y*8),C:PSET(124+X*8,96+Y*
8),C:PSET(124+X*8,92+Y*8),C:PSET(128+X*8,92+Y
*8),C
130 PSET(128+Y*8,96+X*8),C:PSET(124+Y*8,96+X*
8),C:PSET(124+Y*8,92+X*8),C:PSET(128+Y*8,92+X
*8),C
140 PSET(128+X*8,95-Y*8),C:PSET(124+X*8,95-Y*
8),C:PSET(124+X*8,99-Y*8),C:PSET(128+X*8,99-Y
*8),C
150 PSET(128+Y*8,95-X*8),C:PSET(124+Y*8,95-X*
8),C:PSET(124+Y*8,99-X*8),C:PSET(128+Y*8,99-X
*8),C
160 PSET(127-X*8,96+Y*8),C:PSET(131-X*8,96+Y*
8),C:PSET(131-X*8,92+Y*8),C:PSET(127-X*8,92+Y
*8),C
170 PSET(127-Y*8,96+X*8),C:PSET(131-Y*8,96+X*
8),C:PSET(131-Y*8,92+X*8),C:PSET(127-Y*8,92+X
*8),C
180 GOTO 20
```

Auch Commodore arbeitet mit Atari-kompatiblen Joysticks. Allerdings gibt es kein BASIC-Wort, das die Steuerknüppel und Feuerknöpfe liest. Also müssen wir wieder in den *Speicher* hinabsteigen.

Dieses Mal soll aber nicht in den Speicher geschrieben werden (POKE), sondern wir wollen den Inhalt des Speichers *lesen*, nämlich jener Adresse, in der der Zustand des Steuerknüppels als Zahlenwert gehalten wird.

Hierzu dient, wie auch bei Atari und MSX, der Befehl **PEEK**. Der Zustand von Steuerknüppel 1 befindet sich in der Adresse 56337. Mit dem Befehl PEEK(56337) können wir diesen Wert lesen. Welcher Wert in der Speicherzelle steht, ist nicht nur von der Richtung abhängig, sondern auch davon, ob der Feuerknopf gedrückt ist. In Abbildung 10 stehen die Werte in Klammern, die bei gedrücktem Feuerknopf gelesen werden; Joystick 2 beschreibt Adresse 56336.

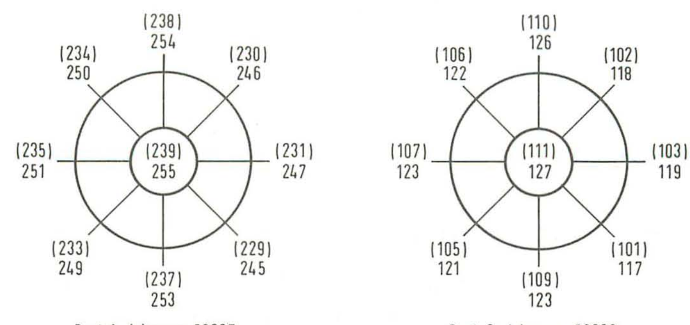


Abb. 10: Die Zuordnung von Zahlenwerten zu Richtungen bei Commodore

Das vollständige Commodore-Programm enthält nun weiter keine Überraschungen mehr:

```
0 REM KALEMOVE.COM
10 T=1024:F=55296:R=RND(-TI):C=1
20 PRINTCHR$(147):POKE 53280,0:POKE 53281,0
30 FOR I=7 TO 31:FOR J=0 TO 24:POKE T+J*40+I,
160:NEXT J:NEXT I:REM 102
40 S=PEEK(56337)
50 IF S<245 THEN C=C+1:IF C>16 THEN C=0
60 IF S=251 THEN X=X-1:IF X<0 THEN X=0
70 IF S=247 THEN X=X+1:IF X>12 THEN X=12
80 IF S=254 THEN Y=Y-1:IF Y<0 THEN Y=0
90 IF S=253 THEN Y=Y+1:IF Y>12 THEN Y=12
100 POKE F+(12-X)*40+19-Y,C
110 POKE F+(12-X)*40+19+Y,C
120 POKE F+(12+X)*40+19+Y,C
130 POKE F+(12+X)*40+19-Y,C
140 POKE F+(12-Y)*40+19-X,C
150 POKE F+(12-Y)*40+19+X,C
160 POKE F+(12+Y)*40+19+X,C
170 POKE F+(12+Y)*40+19-X,C
180 GOTO 40
```

10: Die Basisadressen der beiden Speicherbereiche.

20: Farbe für Rand und Hintergrund.

30: Setzen Sie statt Zeichen 160 wahlweise 102 oder andere!

50: Überprüfen, ob der Feuerknopf gedrückt ist.

60 bis 90: Lesen der Richtung des Steuerknüppels.

100 bis 170: Schreiben der acht Grafikpunkte.

180: Rücksprung.

3.4 Bandwurm

Wie leicht es sein kann »künstliche Intelligenz« vorzugaukeln, und wie wenig dahintersteckt, soll das folgende Programm demonstrieren. Ein Wurm kriecht über den Bildschirm; und als ob er ein lebendiges Wesen wäre, ändert er dann und wann seine Richtung, wandert hin und her, wie ein gefangenes Tier im Käfig.

Wie läßt sich ein solches Programm schreiben? Wir können Grafikpunkte auf den Bildschirm bringen, und wir haben gesehen, wie durch Veränderung der Koordinatenwerte Punkt neben Punkt eine Linie gezogen wird.

Die Bewegungsrichtung ließe sich per Zufall ändern. Da aber die Zufallszahlen statistisch gleichmäßig verteilt fallen, würde der Wurm in einem kleinen Bereich auf der Stelle treten. Wir wollen aber erreichen, daß die Bewegung eine Weile in die gleiche Richtung läuft, sich dann ändert, anders weiterläuft usw.

Sehen wir uns am MSX-Programm einmal an, wie es gemacht wird:

```
0   REM WIRRWURM.MSX
10  SCREEN 3:COLOR 6,1,1:CLS:X=RND(-TIME)
20  X=127:Y=95
100  Q=INT(RND(1)*10)
110  IF Q<9 THEN 200
120  R=INT(RND(1)*4)+1
130  ON R GOTO 150,160,170,180
150  I=4:J=0:GOTO 190
160  I=-4:J=0:GOTO 190
170  I=0:J=4:GOTO 190
180  I=0:J=-4:GOTO 190
190  C=INT(RND(1)*14)+2
200  X=X+I:Y=Y+J
210  IF X<0 THEN X=0:GOTO 120
220  IF X>255 THEN X=255:GOTO 120
230  IF Y<0 THEN Y=0:GOTO 120
240  IF Y>191 THEN Y=191:GOTO 120
250  PSET (X,Y),C
260  GOTO 100
```

20: Die Koordinatenwerte, bei denen der Wurm seine Wanderschaft beginnt.

100: Der Variablen Q wird eine Zufallszahl von 0 bis 9 zugeordnet.

110: Wenn Q kleiner als 9 ist, also in 90% aller Fälle, wird die Programmbearbeitung in Zeile 200 fortgesetzt. Nur etwa jedes zehnte Mal, wenn Q zufällig den Wert 9 bekommen hat, ist diese Bedingung nicht erfüllt, und die Bearbeitung fährt bei Zeile . . .

120 fort, wo die Bewegungsrichtung geändert wird. Dazu wird im ersten Schritt eine Zufallszahl zwischen 1 und 4 gezogen.

130: Diese Zufallszahl bestimmt das Sprungziel. Der ON-GOTO-Befehl verzweigt zur ersten Zeilennummer, wenn die Variable (hier: R) 1 ist, zur zweiten, wenn sie 2 ist usw.

150 bis 180: Der zufällige Sprung führt in eine dieser vier Zeilen, wo den beiden Variablen I und J Werte zugeordnet werden. Soll die Bewegung des Wurmes künftig nach rechts verlaufen, bekommt I einen positiven Wert. Soll die Bewegung nach links verlaufen, bekommt I einen negativen Wert. Mit negativem J geht's nach oben, mit positivem J nach unten. Da bei MSX in SCREEN 3 nur jeder vierte Koordinatenwert einen anderen Grafikpunkt anspricht, beträgt der Wert für I oder J jeweils +4 oder −4.

190: Abschließend wechselt der Wurm seine Farbe.

200: Die Koordinaten X und Y werden um den Wert von I beziehungsweise J verändert. Solange das Programm aus Zeile 110 direkt hierher springt, verändern sich die Koordinaten jedesmal um die gleichen Werte, was auf dem Bildschirm als Bewegung in die gleiche Richtung erscheint. Wurden die Zeilen 110 bis 190 nicht übersprungen, dann haben sich die Werte von I und J verändert, was als Richtungsänderung auf der Mattscheibe erscheint.

210 bis 240 überprüfen die Grenzwerte für die Koordinaten,

250 setzt den neuen Grafikpunkt, und . . .

260 springt für den nächsten Durchlauf zurück nach 100.

Allerdings ist unser kleiner Programmwurm recht dumm. Es geschieht recht häufig, daß er in die Richtung zurückläuft, aus der er gekommen ist. Es kann auch passieren, daß er eine neue Richtung ermittelt, diese aber zufällig mit der alten gleich ist.

Natürlich ließen sich diese Unzulänglichkeiten wegprogrammieren. In den Zeilen 150 bis 180 könnte die zuletzt eingeschlagene Richtung in einer Variablen gemerkt und bei der nächsten Richtungsänderung überprüft werden. Das, was heute so gerne »künstliche Intelligenz« genannt wird, bedeutet nichts anderes, als immer mehr Faktoren zu beobachten und zu berücksichtigen und die Bedingungen für Veränderungen immer weiter zu *differenzieren*, bis sie wie vernünftige Entscheidungen wirken.

Für Atari geschrieben, sieht das gleiche Programm so aus:

```
0 REM WIRRWURM.ATA
10 GRAPHICS 21:POKE 712,0:POKE 708,68:POKE 70
9,148:POKE 710,236
20 X=39:Y=23
100 Q=INT(RND(0)*10)
110 IF Q<9 THEN 200
120 R=INT(RND(0)*4)
130 GOTO 150+R*10
150 I=1:J=0:GOTO 190
160 I=-1:J=0:GOTO 190
170 I=0:J=1:GOTO 190
180 I=0:J=-1:GOTO 190
190 COLOR INT(RND(0)*3)+1
200 X=X+I:Y=Y+J
210 IF X<0 THEN X=0:GOTO 120
220 IF X>79 THEN X=79:GOTO 120
230 IF Y<0 THEN Y=0:GOTO 120
240 IF Y>47 THEN Y=47:GOTO 120
250 PLOT X,Y
260 GOTO 100
```

10: In **GRAPHICS 21** sind die Grafikpunkte so groß wie in SCREEN 3 von MSX. Auch hier erlaubt Atari nur vier Farben gleichzeitig auf der Mattscheibe.

20: Die Startposition für den Bandwurm.

130: Nur Atari erlaubt berechnete Zeilennummern.

150 bis 180: Die Schrittvariablen I und J werden jeweils um +/– 1 verändert.

210 bis 240: Überwachung der Grenzwerte.

Der schlanke Atari-Wurm huscht über die Bildröhre wie vom Blitz getroffen. Vielleicht will er uns zeigen, wie schnell sein Rechner bei der Verarbeitung von Grafikdaten ist?

Der Wurm im Commodore läßt sich nur in der alten Breite programmieren. Dafür können wir ihn aber aus *verschiedenen Grafikzeichen* zusammensetzen. So ist er wenigstens optisch besser in Form als seine Rivalen von MSX und Atari:

```
0 REM WIRRWURM.COM
10 T=1024:F=55296:R=RND(-TI)
20 PRINTCHR$(147):POKE 53280,0:POKE 53281,0
30 FOR J=0 TO 999:POKE T+J,102:NEXT J:REM 91
40 FOR J=0 TO 999:POKE F+J,0:NEXT J
50 X=19:Y=12
100 Q=INT(RND(1)*10)
110 IF Q<9 THEN 200
120 R=INT(RND(1)*4)+1
130 ON R GOTO 150,160,170,180
150 I=1:J=0:GOTO 190
160 I=-1:J=0:GOTO 190
170 I=0:J=1:GOTO 190
180 I=0:J=-1:GOTO 190
190 C=INT(RND(1)*15)+1
200 X=X+I:Y=Y+J
210 IF X<0 THEN X=0:GOTO 120
220 IF X>39 THEN X=39:GOTO 120
230 IF Y<0 THEN Y=0:GOTO 120
240 IF Y>24 THEN Y=24:GOTO 120
250 POKE F+Y*40+X,C
260 GOTO 100
```

30: Experimentieren Sie mit verschiedenen Grafikzeichen; 102 sieht interessant aus, 91 aber nicht minder.

50: Die Startposition.

100 bis 200: Richtungs- und Positionsänderung.

210 bis 240: Grenzwertkontrolle.

250: Die zur neuen Position gehörende Speicherzelle wird mit dem Farbwert C beschrieben. Der gesamte Bildschirm wurde schon in Zeile 30 mit Grafikzeichen gefüllt. Sie werden aber erst sichtbar, wenn hier ein Farbwert in den entsprechenden Farbspeicher geschrieben wird.

260: Rücksprung.

Wie der Wurm so hin und her hastet, wird er immer länger und füllt bald den ganzen Bildschirm aus. Wo er sich gerade herumschlängelt, ist bald nur noch durch die wechselnden Farben zu erkennen. Im folgenden Programm erwecken wir einen »richtigen«, endlichen Wurm zu elektronischem Leben.

3.5 Intelligenzbestie

Unser Wurm bekommt »Augen«! Die braucht er aber auch, denn er ist in einen »Käfig« eingesperrt. Vier kleine Löcher sind in den Wänden. Ob er wohl einen Ausweg findet?

Im folgenden Programm soll der Wurm eine bestimmte Länge haben, nämlich zwanzig Grafikpunkte. Das bedeutet, daß jedesmal, wenn der Wurm vorn einen neuen Grafikpunkt angesetzt bekommt, hinten einer verschwinden soll.

Nun, der ganze Bildschirm ist mit Grafikpunkten ausgefüllt. Verschwinden bedeutet nichts anderes, als daß ein Punkt die *Farbe des Hintergrundes* bekommt und dadurch »unsichtbar« wird. Das größere Problem besteht darin, daß sich unser Würmchen *zufällig* bewegt. Der Computer muß sich also merken, wo sich das Ende des Wurmes befindet. Und das erreichen wir, indem alle zwanzig Glieder, also alle zwanzig Koordinatenwerte, in Variablen aufbewahrt werden:

```
0 REM CAVEWURM.ATA
10 GRAPHICS 21:POKE 712,0:POKE 708,68:POKE 70
9,148:POKE 710,236:DIM X(19),Y(19):FOR I=0 TO
 18:X(I)=0:Y(I)=0:NEXT I
20 X(19)=39:Y(19)=23:C=1
30 COLOR 1:PLOT 24,8:DRAWTO 54,8:DRAWTO 54,38
:DRAWTO 24,38:DRAWTO 24,8
40 COLOR 0:PLOT 34,8:PLOT 43,38:PLOT 24,18:PL
OT 54,28
100 Q=INT(RND(0)*10)
110 IF Q<9 THEN 200
120 R=INT(RND(0)*4)
130 GOTO 150+R*10
150 I=1:J=0:GOTO 190
160 I=-1:J=0:GOTO 190
170 I=0:J=1:GOTO 190
180 I=0:J=-1:GOTO 190
190 C=INT(RND(0)*3)+1
200 X(19)=X(19)+I:Y(19)=Y(19)+J
210 IF X(19)<0 THEN X(19)=79:GOTO 250
220 IF X(19)>79 THEN X(19)=0:GOTO 250
230 IF Y(19)<0 THEN Y(19)=47:GOTO 250
240 IF Y(19)>47 THEN Y(19)=0:GOTO 250
250 LOCATE X(19),Y(19),L:IF L<>0 THEN X(19)=X
(19)-I:Y(19)=Y(19)-J:GOTO 120
260 COLOR C:PLOT X(19),Y(19)
270 COLOR 0:PLOT X(0),Y(0)
280 FOR M=1 TO 19:X(M-1)=X(M):Y(M-1)=Y(M):NEX
T M
300 GOTO 100
```

10: Der Grafikmodus wird eingeschaltet, und Farbwerte werden definiert. Dann werden die Variablen X und Y als Felder mit zwanzig Elementen dimensioniert. Da bei Atari die Elemente dimensionierter numerischer Variablen nicht automatisch den Anfangswert Null haben, kümmert sich die FOR...NEXT-Schleife darum.

20: Die Startposition für unseren Intelligenzwurm und der erste Farbwert.

30: Und das sind seine Gefängnismauern.

40: Vier Ausgänge werden ihm aufgetan.

100 bis 190: Hier wird zufällig die Richtung geändert, wie es beim vorigen Programm schon erläutert wurde.

200 bis 240: Auch die Positionsveränderung erfolgt wie gehabt, nur daß die Koordinaten für den neuen Punkt jetzt jeweils in X(19) und Y(19) verwahrt werden.

250: Das Auge. Bevor dem Wurm ein Schritt erlaubt wird, wirft er einen Blick auf das Feld, das er betreten will. Nur wenn dieses Feld frei ist, also wenn kein Stück Mauer oder ein Glied des Wurmes selbst darauf ist, darf der Schritt getan werden. Der Befehl LOCATE findet bei Atari den Farbwert (COLOR) des Grafikpunktes an der bestimmten Position und ordnet ihn der bestimmten Variablen (hier: L) zu.
Wenn das Feld leer ist, auf das der Wurm gerne treten möchte, dann muß L durch LOCATE den COLOR-Wert 0 (Hintergrund) finden. Wenn (IF) L ungleich 0 ist, dann wird die Änderung von X(19) und Y(19) dadurch rückgängig gemacht, daß die Werte I und J wieder abgezogen werden, die in Zeile 200 addiert wurden. Dann ergeht ein Sprung nach 120, wo eine neue Richtung gesucht wird. Erst wenn der Wurm eine Richtung gewählt hat, in der ein erlaubtes Feld liegt, wird die Bearbeitung des Programms in Zeile

260 fortgesetzt, wo der neue Punkt auf den Bildschirm gebracht wird.

270: Jetzt wird das letzte Glied des Wurmes gelöscht. Seine Koordinaten halten die Variablen X(0) und Y(0). Aber woher haben diese beiden Variablen ihre Werte?

280: Der Wurm »kriecht«, indem alle Koordinatenwerte von der Spitze, X(19),Y(19) um jeweils einen Index nach unten gereicht

werden. Diese Arbeit wird hier durch die FOR...NEXT-Schleife besorgt.

300: Und schon springt das Programm zurück nach 100, wo der nächste Schritt des Wurmes vorbereitet wird.

Wirklich intelligent ist unser Kriechtier natürlich nicht. Es irrt hin und her. Manchmal verharrt es, wenn es nach einer neuen Richtung sucht. Sind wir nicht geneigt zu glauben, es würde nachdenken?

Dann kriecht es weiter, vielleicht direkt auf eine Öffnung zu. Und wir denken, es hat das Loch entdeckt. Aber dann macht das dumme »Weichtier« plötzlich kehrt. Und wir sind entsetzt.

Irgendwann trifft der Wurm natürlich auf eine Lücke in der Mauer und wandert hindurch. Wir atmen erleichtert auf. Doch alles ist nichts als *Zufall.*

Der MSX-Wurm ist auch nicht schlauer. Das Programm enthält nichts Neues:

```
0  REM CAVEWURM.MSX
10 SCREEN 3:COLOR 6,1,1:CLS:X=RND(-TIME):DIM
X(19),Y(19)
20 X(19)=127:Y(19)=95
30 LINE (72,32)-(183,159),14,B
40 PSET(107,32),1:PSET(147,159),1:PSET(72,115
),1:PSET(183,75),1
100 Q=INT(RND(1)*10)
110 IF Q<9 THEN 200
120 R=INT(RND(1)*4)+1
140 ON R GOTO 150,160,170,180
150 I=4:J=0:GOTO 190
160 I=-4:J=0:GOTO 190
170 I=0:J=4:GOTO 190
180 I=0:J=-4:GOTO 190
190 C=INT(RND(1)*14)+2
200 X(19)=X(19)+I:Y(19)=Y(19)+J
210 IF X(19)<0 THEN X(19)=255:GOTO 250
220 IF X(19)>255 THEN X(19)=0:GOTO 250
230 IF Y(19)<0 THEN Y(19)=191:GOTO 250
240 IF Y(19)>191 THEN Y(19)=0:GOTO 250
250 IF POINT (X(19),Y(19))<>1 THEN X(19)=X(19
)-I:Y(19)=Y(19)-J:GOTO 100
260 PSET (X(19),Y(19)),C
270 PSET(X(0),Y(0)),1
280 FOR M=1 TO 19:X(M-1)=X(M):Y(M-1)=Y(M):NEX
T M
300 GOTO 100
```

20: Ausgangsposition.

30: Gefängnismauer.

40: Vier Ausgänge.

100 bis 190: Richtungswahl.

200 bis 240: Die neue Position.

250: Das MSX-Auge heißt **POINT** und findet ebenfalls den Farbwert des bestimmten Grafikpunktes.

260: Der Kopf des Wurmes wandert.

270: Der Schwanz wird gelöscht.

280: Die Variablenwerte kriechen.

300: Rücksprung.

Natürlich könnte man sich eine Menge einfallen lassen, um den Wurm intelligenter zu machen. So wäre es denkbar, die Öffnungen durch Zahlenwerte zu markieren, die das Programm abfragen könnte, wie es jetzt schon mit den freien Feldern getan wird.

Um die Öffnung herum könnte außerdem in abgestuften Werten eine Zone geschaffen werden, die den Wurm veranlaßt, vorrangig in die Richtung der Mauerlücke zu gehen.

Eine andere Möglichkeit bestünde darin, den Gesichtskreis des Wurmes dadurch zu erweitern, daß er nicht nur das Feld überprüft, auf das er ziehen will, sondern einen größeren Bereich von vielleicht drei mal drei Feldern. Wieder müßte der Ausgang irgendwie gekennzeichnet sein, damit das Programm ihn erkennen kann.

So, wie das Programm jetzt geschrieben ist, veranstaltet der Wurm einen Wettlauf gegen seine eigene Dummheit. Denn er kann sich durchaus so winden, daß sein Kopf ringsherum eingeschlossen ist. Dann tritt er ratlos auf der Stelle und findet kein freies Feld, zu dem hin er sich bewegen könnte.

Auch der Commodore-Wurm bringt keine Überraschungen:

```
0 REM CAVEWURM.COM
10 T=1024:F=55296:R=RND(-TI):DIM X(12),Y(12)
20 PRINTCHR$(147):POKE 53280,0:POKE 53281,0
30 FOR J=0 TO 999:POKE T+J,81:POKE F+J,0:NEXT
 J
```

```
40 FOR J=5 TO 19:POKE T+207+J,102:POKE T+767+
J,102:POKE F+207+J,1
50 POKE F+767+J,1:POKE T+J*40+12,102:POKE T+J
*40+26,102:POKE F+J*40+12,1
60 POKE F+J*40+26,1:NEXT J
70 POKE T+217,81:POKE T+426,81:POKE T+572,81:
POKE T+781,81
80 POKE F+217,0:POKE F+426,0:POKE F+572,0:POK
E F+781,0
90 X(12)=19:Y(12)=12:I=0:J=1
100 Q=INT(RND(1)*10)
110 IF Q<9 THEN 200
120 R=INT(RND(1)*4)+1
130 REM ON R GOTO 150,160,170,180
150 IF R=1 THEN I=1:J=0:GOTO 190
160 IF R=2 THEN I=-1:J=0:GOTO 190
170 IF R=3 THEN I=0:J=1:GOTO 190
180 IF R=4 THEN I=0:J=-1:GOTO 190
190 C=INT(RND(1)*14)+2
200 X(12)=X(12)+I:Y(12)=Y(12)+J
210 IF X(12)<0 THEN X(12)=39
220 IF X(12)>39 THEN X(12)=0
230 IF Y(12)<0 THEN Y(12)=24
240 IF Y(12)>24 THEN Y(12)=0
250 IF PEEK(F+Y(12)*40+X(12))<>0 THEN X(12)=X
(12)-I:Y(12)=Y(12)-J:GOTO 100
260 POKE F+Y(12)*40+X(12),C
270 POKE F+Y(0)*40+X(0),0
280 FOR M=1 TO 12:X(M-1)=X(M):Y(M-1)=Y(M):NEX
T M
300 GOTO 100
```

10: Basisadressen.

30: Bildschirmspeicher mit Grafikzeichen 81 füllen.

40 bis 60: Gefängnismauer aus Grafikzeichen 102.

70 und 80: Vier Ausgänge.

90: Anfangswerte.

100 bis 190: Richtungsänderung.

200 bis 240: Ein Schritt.

250: Das Commodore-Auge besteht in einem Blick (PEEK) in den *Speicher*. Dort muß Farbwert 0 (Hintergrund) stehen, damit der Wurm marschieren darf.

260: Der Kopf wird mit Farbe gefüllt und dadurch sichtbar.

270: Der Schwanz wird mit der Hintergrundfarbe (0) gefüllt und dadurch unsichtbar.

300: Rücksprung.

4. Spielplätze

Bislang haben wir dem Rechner nur zugeschaut, wie er allein mit Farben und Formen spielt und dabei eine grafische Flut endlos vor uns ausbreitet. Aus trockenen Zahlen wurden mit ein paar kurzen BASIC-Kommandos farbige und bewegte Bilder.

Der Computer läßt sich aber auch als Spielpartner programmieren, der unsere Züge *registriert* und darauf *reagiert*. Allerdings müssen wir als Programmierer jede mögliche Spielsituation voraussehen und entsprechende Anweisungen für den Rechner formulieren.

Die wohl beeindruckendste Leistung auf diesem Gebiet sind die Schach spielenden Programme. Denn bei der Vielzahl von Zugmöglichkeiten ist es sehr schwierig, ein Programm zu entwerfen, das auf wirklich jede erdenkliche Spielsituation reagieren kann. Dabei ist es gar nicht so besonders aufwendig, die Spielregeln für den Rechner umzuschreiben. Das Problem besteht darin, für jede beliebige Stellung der Figuren auf dem Brett den optimalen Zug zu ermitteln.

In BASIC läßt sich ein brauchbares Schachprogramm ganz bestimmt nicht schreiben. Aber als Spielkamerad hat der Computer mehr zu bieten als die Analyse hochkomplexer Strategiespiele. Schließlich kann er Spielplan, Spieler und Gegner auf den Bildschirm zaubern und damit Spiele ermöglichen, die als konventionelles Brettspiel nicht möglich wären. Er kann Aufgaben stellen und dem Alleinspielenden ein kurzweiliges Gegenüber bieten. Und nicht zuletzt überwacht er, einmal richtig programmiert, unbestechlich und zuverlässig die Einhaltung der Regeln.

4.1 Quizmaster

Im folgenden Spiel stellt der Rechner eine Aufgabe und überwacht die Wertung: Ein Wort aus sechs Buchstaben soll geraten werden. Der Benutzer tippt einen Buchstaben über die Tastatur ein. Kommt dieser Buchstabe im gesuchten Wort vor, so erscheint er auf dem Bildschirm an entsprechender Stelle.

Der Spieler hat nur eine begrenzte Anzahl von Rateversuchen. Nach jedem eingetippten Buchstaben wächst ein Grafikbalken am unteren Bildschirmrand. Ist das Wort nicht erraten, bis der Balken über die gesamte Breite der Mattscheibe reicht, hat der Benutzer verloren.

Wie sich der Rechner eine Zahl ausdenkt, haben wir schon beim Zahlenratespiel kennengelernt. Wie aber kann er sich ein Wort ausdenken? Per Zufall geht das sicherlich nicht, wenn es sich um ein sinnvolles Wort handeln soll.

Wir müssen dem Rechner also eine Menge von Wörtern zur Verfügung stellen, aus der er sich dann willkürlich bedienen kann. Damit ergeben sich für dieses Spielprogramm natürlich gewisse Grenzen, denn irgendwann kennt der Benutzer alle zu ratenden Wörter, und die Aufgabe verliert an Reiz.

Die Programmstruktur des Wortratespiels ähnelt der des Zahlenratens. Hier die MSX-Version:

```
0 REM WORTRATE.MSX
20 SCREEN 1:KEY OFF:R=RND(-TIME):Q=0:V=0
100 W$="BLUMENSCHUNDINGWERMUTTERMINUTERUSSINT
ER"
110 S=INT(RND(1)*12)*3+1
120 R$=MID$(W$,S,6)
130 T$=INKEY$
140 IF T$="" THEN 130
160 FOR P=1 TO 6
170 A$=MID$(R$,P,1)
180 IF A$=T$ THEN LOCATE 10+P,3:PRINT A$:Q=Q+
1
190 NEXT P
200 V=V+1:LOCATE V-1,20:PRINT CHR$(210)
210 IF Q=6 THEN 300
220 IF V=29 THEN 400
230 GOTO 130
300 PRINT:PRINT:PRINT:PRINT:PRINT "--------->
> BRAVO <<---------"
```

```
310 PRINT:PRINT"          Nur";V;"Versuche":GOTO
 500
400 PRINT:PRINT:PRINT:PRINT:PRINT "=======> P
ech gehabt <======="
410 PRINT:GOTO 500
500 PRINT:PRINT:PRINT"            Neues  Spiel?"
510 PRINT:PRINT"          Bitte <J> drücken"
520 IF INKEY$="J" THEN 20
530 GOTO 520
```

20: Der Bildschirmmodus wird gewählt, die Anzeige der Funktionstasten gelöscht, und zwei Variablen werden auf Null gesetzt, was nur bei wiederholtem Durchlauf des Spiels (Rücksprung aus Zeile 520) notwendig ist.

100: Die Variable W$ enthält zwölf Ratewörter, die zu einem Bandwurm aneinandergereiht sind. Alle drei Buchstaben beginnt ein neues Wort. Eine kleine Spielerei am Rande. Natürlich hätte man auch einzelne Wörter in DATA-Zeilen ablegen und per Zufall lesen können. W$ könnte auch länger sein. Vielleicht haben Sie Lust, die Kette zu verlängern, um eine größere Auswahl an Ratewörtern zu ermöglichen.

110: Hier wird durch eine Zufallszahl der Anfangsbuchstabe in W$ gesucht. Es kann eine Zufallszahl von 0 bis 11 gefunden werden. Das entspricht den zwölf möglichen Ratewörtern. Wenn Sie W$ verlängern, kann die Zufallszahl hier entsprechend größer ausfallen. Da nach je 3 Buchstaben in W$ ein Wort beginnt, wird die Zufallszahl mit 3 multipliziert. Und da die Zeichen in einem String, mit 1 beginnend, gezählt werden, die Zufallszahlen, die wir bislang erhalten können, jedoch bei 0 beginnen, muß schließlich noch der Wert 1 addiert werden.

120: Die Variable R$ soll das Ratewort aufnehmen. Dazu müssen wir aus W$ die sechs Zeichen herausholen, die das Wort ergeben. MSX-BASIC hält für solche Aufgaben die Funktion **MID$** bereit. In der Klammer hinter MID$ wird der String angegeben, aus dem Zeichen herausgelesen werden sollen, danach das Zeichen, bei dem begonnen werden soll; das ist in unserem Fall der gerade ermittelte Zufallswert S. Und die Anzahl der Zeichen; das sind sechs.

130: R$ hält jetzt das zu ratende Wort. Nun kommt der Benutzer an die Reihe. Die Funktion **INKEY$** ordnet eine gedrückte Taste der bestimmten Variablen (hier: T$) als String zu.

140: Da der Rechner das Programm schneller abarbeitet, als wir eine Taste drücken können, muß an dieser Stelle eine *Warteschleife* eingebaut werden. Wenn keine Taste gedrückt wurde, hat T$ keinen Wert, das ist der leere String (" ") oder ASCII 0. Dieser Leerstring darf nicht etwa mit der Leerstelle (" ") verwechselt werden, die den ASCII-Wert 32 hat und als Zeichen zu betrachten ist. Wenn also keine Taste gedrückt war, dann wird das Programm nach 130 zurückgeführt.

160: Die FOR...NEXT-Schleife geht die sechs Buchstaben des zu ratenden Wortes einzeln durch, indem sie in Zeile

170 wieder mit der Funktion MID$ einen Buchstaben aus R$ liest und der Variablen A$ zuordnet.

180: Wenn ein Buchstabe (A$) des Ratewortes (R$) mit der gedrückten Taste (T$) gleich ist, dann hat der Benutzer richtig geraten.

Jetzt soll der Buchstabe auf dem Bildschirm gezeigt werden. Das Wort soll in der dritten Zeile erscheinen und in Spalte zehn beginnen. Da der gefundene Buchstabe an P-ter Stelle im Wort steht, muß P zur Spaltenposition addiert werden. LOCATE findet die bestimmte Stelle auf der Mattscheibe, und PRINT gibt den Buchstaben (A$) aus. Die Variable Q übernimmt eine Schiedsrichterfunktion. Sie zählt die Anzahl der geratenen Buchstaben mit.

Wenn das Ratewort den getippten Buchstaben mehrfach enthält, wird er beim Durchlauf der FOR...NEXT-Schleife auch mehrmals gefunden und auf dem Bildschirm angezeigt.

200: Die Variable V zählt die Anzahl der Rateversuche. In der Bildschirmzeile 20 wird in Abhängigkeit von V ein Grafikzeichen (CHR$(210)) ausgegeben. Dadurch wächst mit jedem Versuch am unteren Bildschirmrand ein Balken von links nach rechts.

210: Wenn Q den Wert 6 hat, dann sind alle sechs Buchstaben erraten. Das Spiel ist zu Ende. In Zeile 300 wird das Ergebnis angezeigt.

220: Wenn V den Wert 29 erreicht hat, dann hat der Benutzer verloren. Das Programm fährt in Zeile 400 fort.

300 und 310: Anzeige des Siegergebnisses.

400 und 410: Anzeige des Niederlageergebnisses.

500 und 510: Der Benutzer wird gefragt, ob er noch einmal spielen möchte.

520: Wird die Taste J gedrückt, beginnt das Spiel in Zeile 20 von neuem.

530: Wurde keine oder eine andere Taste gedrückt, wird das Programm nach Zeile 520 zurückgeführt. Wegen dieser Endlosschleife kann das Programm nur durch BREAK beendet werden.

Weil Commodore-BASIC über keinen Befehl verfügt, eine bestimmte Stelle des Bildschirms mit PRINT zu erreichen, müssen wir hier wieder direkt in den Speicher schreiben:

```
0 REM WORTRATE.COM
10 PRINT CHR$(147):POKE 53280,8:POKE 53281,2:
R=RND(-TI):Q=0:V=0:C=1024:F=55296
20 FOR J=0 TO 39:POKE F+200+J,0:NEXT J
30 FOR J=0 TO 39:POKE F+960+J,13:NEXT J
100 W$="BLUMENSCHUNDINGWERMUTTERMINUTERUSSINT
ER"
110 S=INT(RND(1)*12)*3+1
120 R$=MID$(W$,S,6)
130 GET T$
140 IF T$="" THEN 130
150 T=ASC(T$)-64
160 FOR P=1 TO 6
170 A$=MID$(R$,P,1)
180 IF A$=T$ THEN POKE C+216+P,T:Q=Q+1
190 NEXT P
200 V=V+1:POKE C+958+2*V,95:POKE C+959+2*V,10
5
210 IF Q=6 THEN 300
220 IF V=20 THEN 400
230 GOTO 130
300 PRINT:PRINT:PRINT:PRINT:PRINT:PRINT:PRINT
"===============> BRAVO! <==============="
310 PRINT:PRINT"                    NUR";V;"VERSUCHE"
:GOTO 500
400 PRINT:PRINT:PRINT:PRINT:PRINT:PRINT:PRINT
"----------->> PECH GEHABT! <<-----------"
410 PRINT:GOTO 500
500 PRINT:PRINT:PRINT "                  NEUES SP
IEL?"
510 PRINT:PRINT"           BITTE <J> DRUECKEN
"
520 GET T$:IF T$="J" THEN 10
530 GOTO 520
```

10: Bildschirm löschen, Farbe für Rand und Hintergrund, Q und V auf 0 setzen, Startadressen für die beiden Bildschirmspeicherbereiche.

20: In die 40 Spalten der fünften Zeile des Bildschirms wird der Farbwert 0 (Schwarz) geschrieben, damit später hier anzuzeigende Buchstaben entsprechend sichtbar werden können.

30: In die letzte Zeile des Bildschirms wird der Farbwert 13 (Hellgrün) geschrieben; hier soll der Wertungsbalken erscheinen.

100: Der Ratewörterwurm.

110: Die Zufallswahl.

120: Das Ratewort.

130: Bei Commodore wird die Tastatur mit **GET** gelesen.

140: Auch hier muß das Programm bis zur Eingabe in einer Schleife gefangen werden.

150: Um den eingegebenen Buchstaben auf den Bildschirm zu bringen, müssen wir seinen Bildschirmcode in die entsprechenden Speicherzellen schreiben. Im Bereich der Großbuchstaben entspricht der Bildschirmcode des Commodore dem ASCII-Wert des Buchstabens, vermindert um 64. Den ASCII-Wert eines bestimmten Zeichens findet die Funktion **ASC**.

150 bis 190: Die Buchstabeneingabe wird mit dem zu ratenden Wort verglichen, und richtig geratene Buchstaben werden ausgedruckt.

300 bis 530: Diese Zeilen unterscheiden sich vom MSX-Programm nur in geringfügigen Kleinigkeiten.

```
0 REM WORTRATE.ATA
10 DIM W$(40),R$(6),A$(1)
20 GRAPHICS 2:S=0:P=0:Q=0:V=0:POKE 752,1:POKE
 710,32:POKE 764,255
30 FOR C=0 TO 119:POKE 40800+C,0:NEXT C
100 W$="BLUMENSCHUNDINGWERMUTTERMINUTERUSSINT
ER"
110 S=INT(RND(0)*12)*3
120 R$=W$(S+1,S+6)
130 OPEN #1,4,0,"K:"
140 GET #1,T
```

```
150 CLOSE #1
160 FOR P=1 TO 6
170 A$=R$(P,P)
180 IF ASC(A$)=T THEN POSITION 5+P,4:? #6;CHR
$(T):Q=Q+1
190 NEXT P
200 V=V+1:? CHR$(8);CHR$(136);
210 IF Q=6 THEN 300
220 IF V=18 THEN 400
230 GOTO 130
300 ? :? "------------->> BRAVO <<------------
-"
310 ? "            Nur ";V;" Versuche!":GOTO 500
400 ? :? "===========> Pech gehabt! <=========
="
410 ? :GOTO 500
500 ? " Neues Spiel? Bitte <START> tasten:";
510 IF PEEK(53279)=6 THEN 20
520 GOTO 510
```

Bei Atari können wir das Rateergebnis in schönen großen Buchstaben auf dem Bildschirm zeigen. In der Betriebsart GRAPHICS 2 werden die Zeichen in vierfacher Größe und farbig dargestellt:

10: Bei Atari müssen Stringvariablen *grundsätzlich* dimensioniert werden.

20: Grafikmodus und Farbwerte.

30: Alle sechzehn Grafikmodi des Atari können auch mit geteiltem Bildschirm bestimmt werden; am unteren Bildschirmrand stehen dann vier Zeilen für normalen Text (GRAPHICS 0) zur Verfügung. In diesem sogenannten Textfenster ist der POSITION-Befehl aber nicht wirksam. Es ergibt sich also die gleiche Situation wie bei Commodore, daß in den Bildschirmspeicher geschrieben werden muß. Die Basisadresse des Textfenster-Bildschirmspeichers ist 40800.

100: Der Ratewurm.

110: Der Zufallsanfang.

120: Im Gegensatz zu anderen BASIC-Dialekten, die verschiedene String-Befehle verwenden (**LEFT$**, **MID$**, **RIGHT$**), arbeitet Atari mit *Indizes*, die den *ersten* und *letzten Buchstaben* benennen. A$=B$(2,4) bedeutet, daß alle Zeichen vom zweiten bis vierten aus B$ gelesen und A$ zugeordnet werden. A$=B$(6,6) liest nur das sechste Zeichen und ordnet es A$ zu. Da die Zufallszahl S Werte

mit Null beginnend annehmen kann, muß hier zu S 1 addiert werden. Denn auch bei Atari werden die Zeichen eines Strings mit 1 beginnend gezählt.

130 bis 150: Bei Atari gibt es keinen gesonderten Befehl, um die Tastatur zu lesen. Wir müssen erst einen Datenkanal zur Tastatur öffnen, dann aus dem Datenkanal mit GET ein Datum lesen (das ist der ASCII-Code der gedrückten Taste) und dürfen nicht vergessen, den Datenkanal auch wieder zu schließen.

160: Die sechs Buchstaben des zu ratenden Wortes werden einzeln durchgegangen,

170: mit der gedrückten Taste verglichen,

180: und richtige Buchstaben werden auf den Bildschirm gebracht.

200 bis 500: Keine grundsätzlichen Unterschiede zu MSX oder Commodore.

510: In der Adresse 53279 wird der Zustand der drei Funktionstasten OPTION, SELECT und START gehalten. In diesem Register findet PEEK den Wert 6, wenn die START-Taste gedrückt ist.

4.2 Suchanzeige

Wir befinden uns in einem *interstellaren Raumschiff*, und der einzige Kontakt zur Außenwelt besteht in einem Ortungsgerät, das die Entfernung zu unserer Heimatbasis anzeigt.

Das Raumschiff wird über die Tastatur unseres Heimcomputers in beliebige Richtungen gesteuert, nach oben oder unten, nach rechts oder links, nach vorn oder hinten. Das Ortungsgerät zeigt jedoch immer nur die direkte Entfernung an, nicht aber die Richtung. Obendrein *bewegt sich die Heimatbasis*, so daß es gar nicht so leicht ist, die Entfernung bis auf Null zu reduzieren.

Das Atari-Programm:

```
0 REM ORTUNGEN.ATA
10 GRAPHICS 7:POKE 712,0:POKE 708,34:POKE 709
,178:POKE 710,42:C=1:POKE 752,1
20 FOR J=0 TO 36
```

```
30 C=C+1:IF C>2 THEN C=1
40 COLOR C:PLOT J,J:DRAWTO 159-J,J:DRAWTO 159
-J,79-J:DRAWTO J,79-J:DRAWTO J,J
50 NEXT J
60 FOR J=37 TO 42:COLOR 0:PLOT 0,J:DRAWTO 159
,J:NEXT J
70 ? :? "<O> OBEN      <L> LINKS    <V> VORN"
80 ? "<U> UNTEN    <R> RECHTS   <H> HINTEN";
100 XC=INT(RND(0)*47)
110 YC=INT(RND(0)*47)
120 ZC=INT(RND(0)*47)
130 XU=INT(RND(0)*47)
140 YU=INT(RND(0)*47)
150 ZU=INT(RND(0)*47)
200 V=V+1
210 XD=ABS(XC-XU)
220 YD=ABS(YC-YU)
230 ZD=ABS(ZC-ZU)
240 D=INT((((XD^2+YD^2)^0.5)^2+ZD^2)^0.5)
250 COLOR 3:PLOT 0,39:DRAWTO 159,39
260 PLOT 0,40:DRAWTO 159,40
270 COLOR 1:PLOT 79-D,39:DRAWTO 79+D,39
280 PLOT 79-D,40:DRAWTO 79+D,40
290 IF D=0 THEN 500
300 OPEN #1,4,0,"K:":GET #1,T:CLOSE #1
310 IF T=72 THEN ZU=ZU-2:IF ZU<0 THEN ZU=O:?
CHR$(253);
320 IF T=76 THEN XU=XU-2:IF XU<0 THEN XU=O:?
CHR$(253);
330 IF T=79 THEN YU=YU+2:IF YU>46 THEN YU=46:
? CHR$(253);
340 IF T=82 THEN XU=XU+2:IF XU>46 THEN XU=46:
? CHR$(253);
350 IF T=85 THEN YU=YU-2:IF YU<0 THEN YU=O:?
CHR$(253);
360 IF T=86 THEN ZU=ZU+1:IF ZU>46 THEN ZU=46:
? CHR$(253);
400 Q=INT(RND(0)*6)
410 GOTO 420+Q*10
420 XC=XC-0.5:IF XC<0 THEN XC=46
425 GOTO 200
430 YC=YC-0.5:IF YC<0 THEN YC=46
435 GOTO 200
440 ZC=ZC-0.5:IF ZC<0 THEN ZC=46
445 GOTO 200
450 XC=XC+0.5:IF XC>46 THEN XC=0
455 GOTO 200
460 YC=YC+0.5:IF YC>46 THEN YC=0
465 GOTO 200
470 ZC=ZC+0.5:IF ZC>46 THEN ZC=0
475 GOTO 200
500 COLOR 3:PLOT 0,39:DRAWTO 159,39
510 PLOT 0,40:DRAWTO 159,40
520 ? :? :? :? "                GEORTET!"
530 ? :? "  Sie haben ";V;" Schritte gebrauch
t."
```

10: GRAPHICS 7 ist eine sehr feine Auflösung, jeder Grafikpunkt ist nur zwei Bildschirmzeilen hoch und zwei Bildpunkte breit. Auch in dieser Betriebsart kann Atari vier Farben gleichzeitig darstellen. Die Farbwerte dafür werden hier in die Farbregister geschrieben.

20 bis 60 malen einen Hintergrund, wie es nur Atari kann.

70 und 80 schreiben die Tastenfunktionen in das Textfenster, während im oberen Bildschirmbereich Grafik gezeigt wird. Auch diese Möglichkeit ist bei Atari einzigartig einfach.

100 bis 120: Die Suche spielt sich in einem Raum von 47 mal 47 mal 47 Feldern ab. Die Position der Heimatbasis wird durch die Variablen XC, YC und ZC bestimmt. Hier werden zufällige Werte dafür gefunden.

130 bis 150: Ebenso werden zufällige Werte für die Position unseres Raumschiffs ermittelt.

200: Das Spiel beginnt. Die Anzahl der Schritte des Benutzers wird in den Variablen V gezählt.

210 bis 230: Hier wird die Entfernung des Raumschiffs zur Heimatbasis in den drei einzelnen Dimensionen ermittelt. Die Entfernung entspricht der Differenz der jeweiligen Werte. Da eine Entfernung immer positiv ist, unterdrücken wir mit der Funktion **ABS** das Vorzeichen.

240: Diese Formel ermittelt nach dem Satz des Pythagoras aufgrund der Differenzen der Koordinatenwerte aus den Zeilen 210 bis 230 die direkte Entfernung; Nachkommastellen werden durch **INT** unterdrückt, denn der gefundene Wert D wird für die Darstellung der Entfernung verwendet.

250 und 260: Mit COLOR 3 wird eine neutrale Linie in voller Länge gezeichnet.

270 und 280: Dann wird mit Farbe 1 eine Linie gezeichnet, deren Länge von D abhängt.

290: Wenn die Heimatbasis erreicht ist, also wenn der Abstand D den Wert 0 erreicht hat, ist die Suche beendet. Ab Zeile 500 wird dann das Ergebnis angezeigt.

300: Die Tastatur wird abgefragt und die gedrückte Taste in T erfaßt.

310 bis 360: Je nachdem, welche Taste gedrückt wurde, wird die entsprechende Variable für die Position unseres Raumschiffs vergrößert oder verkleinert. Als Grenzwerte werden 0 und 46 überprüft. Da das Raumschiff dann am Rande unseres Spielraums festsitzt, wird mit PRINT CHR$(253) ein Signalton abgegeben, der dem Benutzer sagt, daß es in dieser Richtung nicht weitergeht.

400 bis 475: Die Heimatbasis macht einen zufälligen Zug, wie es beim WIRRWURM-Programm bereits beschrieben wurde. Wenn die Heimatbasis einen Grenzwert erreicht, springt sie zum gegenüberliegenden Rand hinüber. Der Rücksprung nach Zeile 200 leitet den nächsten Zug ein.

500 bis 530 signalisieren das Ende des Spiels und drucken das Ergebnis auf den Bildschirm.

Für Commodore läßt sich dieses Programm von BASIC aus nicht sinnvoll programmieren, weil die Darstellung der Entfernung als Balken zu grob ausfallen würde. Man müßte hier wohl einen ganz anderen Weg gehen und die Entfernung z. B. numerisch darstellen.

Auch für MSX läßt sich dieses Programm nur mit gewissen Einschränkungen schreiben. Die Darstellung der Entfernung als grafischer Balken fällt etwas gröber aus. Dadurch wird die Orientierung für den Spieler reichlich erschwert. In der hier abgedruckten Version wurde deshalb darauf verzichtet, die Heimatbasis sich bewegen zu lassen, weil ein Andocken damit fast unmöglich wird. Aber probieren Sie es selbst aus. Setzen Sie in den Zeilen 420 bis 470 statt der 0 einen (kleinen) Wert ein:

```
0  REM ORTUNGEN.MSX'
10 SCREEN 3:COLOR 14,1,1:CLS:R=RND(-TIME)
20 FOR J=0 TO 21
30 CC=CC+1:IF CC>1 THEN CC=0
40 C=6+CC*6:I=J*4
50 LINE (0+I,0+I)-(255-I,191-I),C,B
60 NEXT J
70 FOR J=0 TO 4:LINE (0,88+J*4)-(255,88+J*4),
1:NEXT J
100 XC=INT(RND(1)*40)
110 YC=INT(RND(1)*40)
120 ZC=INT(RND(1)*40)
```

```
130 XU=INT(RND(1)*40)
140 YU=INT(RND(1)*40)
150 ZU=INT(RND(1)*40)
200 V=V+1:DA=DN
210 XD=ABS(XC-XU)
220 YD=ABS(YC-YU)
230 ZD=ABS(ZC-ZU)
240 DN=(((XD^2+YD^2)^.5)^2+ZD^2)^.5:D=INT(DN)
250 IF DA<=DN THEN B=13
260 IF DA>DN THEN B=4
270 LINE (0,96)-(255,96),B
280 LINE (127-D*2,96)-(128+D*2,96),11
290 IF D=0 THEN 500
300 T$=INKEY$:K=0
310 IF T$="H"THEN K=1:ZU=ZU-1:IF ZU<0 THEN ZU
=0:PRINT CHR$(7)
320 IF T$="L"THEN K=1:XU=XU-1:IF XU<0 THEN XU
=0:PRINT CHR$(7)
330 IF T$="O"THEN K=1:YU=YU+1:IF YU>39 THEN Y
U=39:PRINT CHR$(7)
340 IF T$="R"THEN K=1:XU=XU+1:IF XU>39 THEN X
U=39:PRINT CHR$(7)
350 IF T$="U"THEN K=1:YU=YU-1:IF YU<0 THEN YU
=0:PRINT CHR$(7)
360 IF T$="V"THEN K=1:ZU=ZU+1:IF ZU>39 THEN Z
U=39:PRINT CHR$(7)
370 IF K=0 THEN 300
400 Q=INT(RND(1)*6)+1
410 ON Q GOTO 420,430,440,450,460,470
420 XC=XC-0:IF XC<0 THEN XC=39
425 GOTO 200
430 YC=YC-0:IF YC<0 THEN YC=39
435 GOTO 200
440 ZC=ZC-0:IF ZC<0 THEN ZC=39
445 GOTO 200
450 XC=XC+0:IF XC>39 THEN XC=0
455 GOTO 200
460 YC=YC+0:IF YC>39 THEN YC=0
465 GOTO 200
470 ZC=ZC+0:IF ZC>39 THEN ZC=0
475 GOTO 200
500 LINE (0,96)-(255,96),14
510 OPEN "GRP:" FOR OUTPUT AS #1
520 COLOR 4,1,1
530 PSET (12,8),6:PRINT #1,"GEORTET!"
540 PSET (0,159),6:PRINT #1,V
550 PSET (127,159),6:PRINT #1,"ZÜGE"

600 GOTO 600
```

10: Grafikmodus, Farbe für Hintergrund und Rand, Anfangswert für Zufallszahlen.

20 bis 70: Durch die gröbere Auflösung fällt die Hintergrundgrafik etwas schlichter aus.

100 bis 120: Zufällige Startkoordinaten für die Heimatbasis.

130 bis 150: Zufällige Startkoordinaten für unser Raumschiff.

200: Die Versuche werden in V gezählt. Der Variablen DA wird der Abstand zwischen Raumschiff und Basis aus der letzten Runde zugeordnet.

210 bis 230: Die absoluten Differenzen der Koordinatenwerte werden ermittelt.

240: Mit der gleichen Formel wie bei Atari wird der Abstand zwischen Heimatbasis und Raumschiff berechnet.

250 und 260: Da die Orientierung durch die gröbere Auflösung erschwert ist, gibt es hier eine zusätzliche Hilfe. Der Hintergrund des Entfernungsbalkens wechselt seine Farbe. War der vorherige Abstand kleiner oder gleich dem jetzigen Abstand, wenn also der Spieler nicht vorangekommen ist, dann wird der Entfernungsbalken mit Farbe 13 (Magentarot) unterlegt: Gefahr, wir entfernen uns! War der alte Abstand größer, dann signalisiert Farbe 4 (Dunkelblau), daß alles in Ordnung ist.

270: Der Hintergrund für den Entfernungsbalken wird gezeichnet.

280: Der Entfernungsbalken selbst wird in Farbe 11 (Hellgelb) dargestellt.

290: Wenn die Endbedingung erfüllt ist, Abstand D gleich Null, wird das Ergebnis ab Zeile 500 ausgegeben.

300: Die Funktion INKEY$ fragt wieder die Tastatur ab, und die gedrückte Taste wird der Variablen T$ zugeordnet. Da wir wieder irgendwie überprüfen müssen, ob bereits eine Taste gedrückt wurde, gleichzeitig aber auch sichergestellt werden muß, daß es sich nicht nur um irgendeine Taste handelt, wird die Betätigung einer zulässigen Taste mit der Variablen K kontrolliert. Hier wird K auf Null gesetzt.

310 bis 360: Enthält T$ das Zeichen einer zulässigen Taste, wird die Kontrollvariable auf 1 gesetzt und die entsprechende Koordinatenvariable verändert. Wenn ein Grenzwert erreicht ist, löst **PRINT CHR$(7)** einen Kontrollton aus.

400 bis 475: Die Zufallszahl Q bestimmt die Richtung, in der die Heimatbasis vom Rechner verschoben wird. Im abgedruckten

Listing ist allerdings die Veränderung 0 bestimmt, die Heimatbasis ist also stationär, um das Spiel etwas zu erleichtern.

480: Rücksprung für den nächsten Zug.

500: Der Entfernungsbalken wird gelöscht.

510: Ein Datenkanal zum Grafikbildschirm wird geöffnet.

520: Farbwerte werden bestimmt.

530: An die Position 12,8 wird ein Grafikpunkt gesetzt. Dann wird über den geöffneten Datenkanal #1 eine Stringkonstante ausgegeben. Auf diese Weise eröffnet MSX einen recht bequemen Weg, um in einen Grafikbildschirm hinein Schriftzeichen zu setzen, was sonst nur schwer oder mit Einschränkungen möglich ist.

540 und 550 beschreiben auf entsprechende Weise den Grafikbildschirm.

600: Diese Zeile verhindert die Beendigung des Programmes. Das System würde sonst den Textmodus SCREEN 0 einschalten, um die Eingabebereitschaft anzuzeigen. Das hätte aber zur Folge, daß die bestehende Grafik gelöscht würde.

4.3 Irrführer

Wenn mit dem Computer ein Spielfeld auf dem Monitor dargestellt wird, dann handelt es sich oft um ein *Labyrinth*, weil so auf der doch relativ kleinen Bildfläche recht lange Wege und viel Spielraum geschaffen werden können.

Ein Irrgarten besteht aus verschlungenen Pfaden und undurchdringlichen Mauern. Wie der Rechner zwischen erlaubten und verbotenen Feldern unterscheiden kann, haben wir schon bei den Programmen WIRRWURM und CAVEWURM kennengelernt.

Bei diesem Spiel wollen wir uns nun nicht mit einem immer gleichen Labyrinth zufriedengeben. Denn das Spiel wäre nach zwei, drei Versuchen langweilig. Unser Spielprogramm soll bei jedem Mal einen anderen Irrgarten vor uns ausbreiten, denn auch Labyrinthe lassen sich zufällig erzeugen.

Die komplizierten Überlegungen wollen wir uns jedoch ersparen, die es ermöglichen, ein Labyrinth im üblichen Sinne per Zufall vom

Rechner zeichnen zu lassen; ein Labyrinth also, durch das ein und nur ein Weg hindurchführt.

Wir machen uns die Sache einfacher und verteilen Wege und Mauern willkürlich über die Bildfläche. Dabei kommt es natürlich sehr auf das *richtige Mischungsverhältnis* an. Wenn das Programm zu viele Mauern setzt, dann führt womöglich gar kein Weg hindurch; werden zu wenige Mauern errichtet, dann macht das Spiel keinen Spaß, weil die Wege offenliegen.

Sie denken, eine zufällige Verteilung von Mauern und Wegen ist immer leicht zu durchqueren, selbst wenn das Mischungsverhältnis fein abgestimmt ist? Das mag schon sein. Aber in unserem Spiel werden weder Wege noch Mauern zu sehen sein!

Der Spieler beginnt als Grafikpunkt in der Mitte des linken Bildschirmrandes. Vor ihm liegt nichts als eine schwarze Fläche. Mit dem *Joystick* steuert er sich über die Spielfläche und hinterläßt Grafikpunkte, die seinen Weg sichtbar machen. Erst wenn er mit dem Kopf gegen eine Wand läuft, wird diese sichtbar.

Ziel ist, den rechten Bildschirmrand zu überschreiten. In diesem Moment wird das gesamte Spielfeld erleuchtet, und alle Mauern werden sichtbar. So kann der Benutzer abschließend den Irrweg betrachten, den er im Dunkeln ertastet hat.

Natürlich gibt es eine gewisse Möglichkeit, daß gar keine Bahn von links nach rechts frei ist, aber es ist sehr unwahrscheinlich und kommt beim vorliegenden Programm in der Praxis eigentlich nicht vor. Wenn Sie aber in Zeile 60 das Mischungsverhältnis von Wegen und Mauern verändern, indem Sie statt >.69 vielleicht >.60 oder andere Werte eingeben, können Sie ausprobieren, wo der kritische Punkt für unbegehbare Verteilungen liegt.

Zuerst das Programm LABYRINT.COM, also die Commodore-Fassung:

```
0 REM LABYRINT.COM
10 T=1024:F=55296:R=RND(-TI)
20 PRINTCHR$(147):POKE 53280,0:POKE 53281,0
30 FOR J=0 TO 24
40 FOR I=0 TO 39
50 POKE T+960+I,J+1:POKE F+J*40+I,0
```

```
60 Z=81:IF RND(1)>.69 THEN Z=86:REM 81=WEG -
86 = MAUER
70 POKE T+J*40+I,Z
80 NEXT I
90 NEXT J
100 POKE T+440,81:POKE T+480,81:POKE T+481,81
:POKE T+520,81
210 X=0:Y=12
220 POKE F+(Y+V)*40+(X+U),2
230 U=0:V=0
240 POKE F+Y*40+X,7
250 S=PEEK(56337):REM JOYSTICK PORT 1
260 IF S=255 THEN 250
270 IF S=247 THEN X=X+1:U=-1
280 IF X>39 THEN 400
290 IF S=251 THEN X=X-1:U=1
300 IF X<0 THEN X=0:U=0
310 IF S=253 THEN Y=Y+1:V=-1
320 IF Y>24 THEN Y=24:V=0
330 IF S=254 THEN Y=Y-1:V=1
340 IF Y<0 THEN Y=0:V=0
350 IF PEEK (T+Y*40+X)=81 THEN 220
360 POKE F+Y*40+X,9
370 X=X+U:Y=Y+V
380 GOTO 220
400 FOR Y=0 TO 24
410 FOR X=0 TO 39
420 IF PEEK(T+Y*40+X)=86 THEN POKE F+Y*40+X,1
1
430 NEXT X
440 NEXT Y
500 B=54272
510 FOR Y=1 TO 3
520 L=10
530 FOR H=70 TO 100 STEP 2
540 POKE B+24,L:POKE B+6,15*16:POKE B+1,H:POK
E B+4,33
550 L=L+1:IF L>14 THEN L=15
560 NEXT H
570 NEXT Y
580 FOR W=0 TO 50:NEXT W:GOTO 500

READY.
```

10: Die Basisadressen der beiden Bildschirmspeicherbereiche werden in den Variablen erfaßt; mit der Timer-Variablen wird ein Anfangswert für die Pseudozufallszahlenreihe gesetzt.

20: Der Bildschirm(speicher) wird gelöscht; Farben für Rand und Hintergrund auf 0 (Schwarz) gesetzt.

30: Jetzt wird das zufällige Labyrinth erzeugt. Die doppelte FOR-...NEXT-Schleife geht mit den Variablen J die 25 Zeilen und

40 mit I die 40 Spalten des Bildschirms durch.

50: Es dauert eine kleine Weile, bis für eintausend Bildschirmpositionen zufällige Werte in den Speicher geschrieben sind. Selbst ein paar Sekunden sind aber für den Betrachter eine unangenehm lange Zeit, wenn auf dem Bildschirm nichts zu sehen ist. Hier wird deshalb die unterste Zeile des Bildschirms, die zuletzt mit Labyrinthelementen gefüllt wird, fortlaufend mit Zeichen des Alphabets beschrieben.

Die erste Speicherzelle der letzten Bildschirmzeile beginnt bei T+960. I geht also jeweils die 40 Spalten dieser Zeile durch, während der Wert J+1 hineingeschrieben wird. Beim ersten Durchlauf hat J den Wert 0, die unterste Zeile wird also mit dem Bildschirmcode 1 beschrieben, das ist das »A«; beim nächsten Durchlauf hat J den Wert 1, die Zeile wird mit »B« beschrieben etc. Natürlich könnte man mit etwas mehr Aufwand auch sinnvollere Ereignisse in der Wartezeit ermöglichen. So könnte, statt das Alphabet durchzugehen, auch die Nachricht »Bitte 30 Sekunden Geduld« als Laufschrift gezeigt werden.
Das zweite Statement der Zeile beschreibt den Farbwertespeicher mit 0 (Schwarz).

60: Die Variable Z wird auf 81 gesetzt. Der Bildschirmcode 81 gehört zu dem dicken Punkt, der bei diesem Spiel den Weg anzeigen soll. Wenn eine gezogene Zufallszahl größer als 0,69 ist, also in 31% der Fälle, wird Z aber auf 86 gesetzt. Der Bildschirmcode 86 erzeugt ein Kreuz, das die Mauer markieren soll.

70: Der Wert Z wird in den Zeichenspeicher geschrieben. Da der Farbwert (0) aber identisch mit dem Farbwert des Hintergrundes ist, erscheint auf dem Bildschirm nichts; das Labyrinth ist unsichtbar.

80: Ende der inneren Schleife.

90: Ende der äußeren Schleife.

100: Da es relativ leicht geschehen kann, daß die drei Felder rings um die Startposition des Spielers mit Mauern verbaut werden, der Spieler also keinen einzigen Schritt machen kann, werden diese drei Felder hier vorsorglich gesondert zu Wegen bereitet.

210: Die Koordinaten der Startposition werden in den Variablen X und Y erfaßt.

220: Der Spieler erscheint als rundes Grafikzeichen in Farbe 2 (Rot) auf der Startposition.

230: Die Variablen U und V werden auf 0 gesetzt, um sie für den (neuen) Durchlauf vorzubereiten.

240: Die Position des Spielers wird mit Farbe 7 (Gelb) markiert. Das ist wichtig, damit sich der Benutzer auf dem Bildschirm orientieren kann. Wenn er sich bewegt, zeichnet er zwar eine rote Spur auf den schwarzen Bildschirm. Muß er aber einen Weg zurückgehen, der schon rot gefärbt ist, dann wäre die aktuelle Position nicht mehr auszumachen, wenn sie nicht in einer abweichenden Farbe gezeigt würde.

250: Das Register des Joysticks wird in bekannter Weise abgefragt.

260: Wenn der Joystick in Ruhestellung ist, wird sofort zur erneuten Abfrage verzweigt.

270: Wenn der Steuerknüppel nach rechts gedrückt ist, wird die X-Koordinate um 1 erhöht. In U wird der entgegengesetzte Wert (−1) abgelegt.

280: Wenn X größer als 39 ist, dann wurde der rechte Bildschirmrand überschritten. Die Siegbedingung ist erfüllt; das Programm verzweigt nach Zeile 400.

290: Wurde der Steuerknüppel nach links gedrückt, werden der X- und U-Wert entsprechend verändert; Zeile

300 überprüft den Grenzwert.

310 bis 340: Auf gleiche Weise wird mit der Bewegung nach oben und unten, Y-Koordinate, verfahren.

350: Jetzt sehen wir in der Speicherzelle nach, die der neu ermittelten Bildschirmposition zugeordnet ist. Wenn dort der Code eines Wegstückes (81) liegt, dann wird die Bearbeitung mit Zeile 220 fortgesetzt, wo die alte Position (X+U,Y+V) mit Rot (2) beschrieben wird und später, in Zeile 240, die neue Position mit Gelb.

360: Sind wir aber mit dem Kopf gegen eine Wand gelaufen, dann wird mit Farbe 9 (Braun) das Kreuz sichbar gemacht, das die Mauer darstellt.

370: Mit Hilfe der Variablen U und V werden die alten Koordinatenwerte des Spielers wiederhergestellt, damit er nicht mit dem Kopf durch die Wand hindurchstößt.

380: Dann erfolgt der Rücksprung nach Zeile 220, wo ein neuer Durchlauf beginnt, der Spieler den nächsten Schritt macht.

400 bis 440: War in Zeile 280 die Siegbedingung erfüllt, dann ist das Programm hierher gelangt. Wie in den Zeilen 30 bis 90 werden mit einer doppelten FOR...NEXT-Schleife alle eintausend Bildschirmpositionen gelesen. Und wenn die Zelle im Zeichenspeicher den Wert 86 (Mauer) hält, dann wird in die entsprechende Adresse des Farbwertespeichers eine 1 (Weiß) geschrieben. Die Labyrinthmauern werden Zeile um Zeile sichtbar.

500 bis 580: Diese Zeilen erzeugen einen Heulton, der endlos wiederholt wird, bis der Benutzer das Programm abbricht. Auch für die Tonerzeugung hat Commodore keine BASIC-Befehle. Die Anweisungen müssen direkt in den entsprechenden Adressen abgelegt werden, die vom Tongenerator gelesen werden.

Da Commodore die Inhalte des Bildschirms auf zwei Ebenen speichert, einmal die Zeichencodes, zum anderen die Farbwerte, konnten wir das Labyrinth im Zeichenspeicher ablegen, ohne daß es sichtbar wurde.

Da bei Atari alle Daten für den Bildschirm in einem Speicherbereich erfaßt sind, müssen wir hier einen anderen Weg finden, um das gleiche zu erreichen. Wir benötigen eine sichtbare Ebene, auf der Wege, Mauern und Spieler im aktuellen Stand gezeigt werden, und eine unsichtbare Ebene, auf der das gesamte Labyrinth erfaßt ist.

Die zweite, unsichtbare Ebene liefert uns eine doppeltindizierte Variable, die so viele Elemente hat, wie der Bildschirm Grafikpunkte faßt.

```
0 REM LABYRINT.ATA
10 DIM F(39,23):POKE 82,0:POKE 752,1
20 ? CHR$(125):? "****************************
*************";
30 ? "*        BITTE 30 SEKUNDEN GEDULD
*****************************************":?
:? :?
```

```
40 FOR I=0 TO 39:? CHR$(20);
50 FOR J=0 TO 23
60 C=0:IF RND(0)>0.69 THEN C=3
70 F(I,J)=C
80 NEXT J
90 NEXT I
100 F(1,11)=0:F(0,10)=0:F(0,12)=0
200 GRAPHICS 19:POKE 710,2:POKE 708,50
210 X=0:Y=11
220 COLOR 1:PLOT X+U,Y+V
230 U=0:V=0
240 COLOR 2:PLOT X,Y
250 S=STICK(0)
260 IF S=15 THEN 240
270 IF S=7 THEN X=X+1:U=-1
280 IF X>39 THEN 400
290 IF S=11 THEN X=X-1:U=1
300 IF X<0 THEN X=0:U=0
310 IF S=13 THEN Y=Y+1:V=-1
320 IF Y>23 THEN Y=23:V=0
330 IF S=14 THEN Y=Y-1:V=1
340 IF Y<0 THEN Y=0:V=0
350 IF F(X,Y)=0 THEN GOTO 220
360 COLOR 3:PLOT X,Y
370 X=X+U:Y=Y+V
380 GOTO 220
400 FOR Y=0 TO 23
410 FOR X=0 TO 39
420 IF F(X,Y)=3 THEN COLOR 3:PLOT X,Y
430 NEXT X
440 NEXT Y
450 FOR T=12 TO 144 STEP 12:SOUND 0,T,14,10:N
EXT T:GOTO 450
```

10: Die doppeltindizierte Variable muß dimensioniert werden. Der linke Bildschirmrand wird auf 0 gesetzt und die Anzeige des Cursors unterdrückt.

20: Dann wird der Bildschirm gelöscht und ein Wartehinweis ausgegeben.

40 bis 90: Wie beim Commodore-Programm die Bildschirmfläche Spalte um Spalte und Zeile um Zeile gefüllt wurde, wird hier die doppeltindizierte Variable mit Werten belegt.
Die Beziehung der Variablen F(n,n) zum Bildschirm wird später so betrachtet, daß das Element F(0,0) dem Grafikpunkt in der linken oberen Ecke zugeordnet ist, F(39,0) der rechten oberen Ecke entspricht, F(39,23) zur Ecke unten rechts und F(0,23) zur Ecke unten links gehört. Die beiden Indizes der Variablen werden also behandelt, als ob sie die Koordinaten einer Bildschirmposition wären.

Jedesmal wenn die doppelte FOR...NEXT-Schleife eine neue Spalte zu bearbeiten beginnt, wird auf den Bildschirm das Zeichen 20 gedruckt, das bei Atari dem Blockgrafikzeichen dicker Punkt entspricht. So hat der Betrachter durch einen wachsenden Balken aus Punkten eine Kontrolle über das Verrinnen der Wartezeit.

In den 40 mal 24 Elementen der doppeltindizierten Variablen F wird der Wert 0 für ein Wegstück gefaßt oder, wenn es der Zufall will, der Wert 3 für Mauer.

100: Die drei Elemente rings um die Startposition werden zu Wegen geebnet.

200: **GRAPHICS 19** entspricht der Auflösung von GRAPHICS 3, nur daß am unteren Bildschirmrand keine vier Zeilen Textfenster dargestellt werden. Die Grafikpunkte sind acht Bildschirmzeilen hoch und acht Bildpunkte breit. Die gesamte Fläche ist in 40 mal 24 Grafikpunkte aufgelöst, also genau wie im Textbetrieb (GRAPHICS 0), nur daß in den 960 Schreibstellen keine einfarbigen Zeichen, sondern Grafikpunkte in vier verschiedenen Farben dargestellt werden.

210: Anfangskoordinaten für den Spieler.

220: Mit Farbe 1 aus Farbregister 708 wird die alte Position des Spielers gezeichnet. Im Farbregister 708 wurde der Wert 50 abgelegt, so daß COLOR 1 einen roten Farbton aufruft.

230: Die Variablen U und V werden auf 0 gesetzt.

240: COLOR 2 bezieht sich auf Farbregister 709, für das wir keinen Wert bestimmt haben, so daß der Grundwert 202 (Hellgrün) aktiv bleibt. Die neue Spielerposition wird hier mit Farbe 2 gezeichnet.

250 bis 340 besorgen die Abfrage des Steuerknüppels und die entsprechenden Veränderungen der Variablen, wie für die Commodore-Version erklärt.

350: Auf der unsichtbaren Ebene der doppeltindizierten Variablen wird nun nachgesehen, ob das neue Feld ein Weg oder eine Mauer ist. Wenn auf der neuen Position der Wert 0 (=Weg) gehalten wird, läuft das Programm bei Zeile 220 weiter.

360: Sonst gelangt das Programm in diese Zeile, wo die neue Position mit Farbe 3 aus Register 710 gezeichnet wird. Ein hellgrauer Grafikpunkt wird als Mauer gesetzt.

370: Die Koordinatenvariablen X und Y werden auf den alten Wert zurückgebracht. Der Spieler hat sich nicht bewegt, nur das Mauerstück ist vor seiner Nase erschienen.

380: Rücksprung zum nächsten Durchlauf.

400 bis 440: Wurde in Zeile 280 die Siegbedingung erfüllt, werden die 960 Elemente der Variablen F durchgesehen. Immer wenn F(n,n) den Wert 3 hat, also wenn ein Mauerstück vorliegt, wird der entsprechende Grafikpunkt mit COLOR 3 sichtbar gemacht.

450: Hier erzeugt der SOUND-Befehl ein Geräusch, das durch Rücksprung in die eigene Zeile zu einem Dauerereignis wird.

Für MSX sieht das gleiche Programm im wesentlichen so aus wie die Atari-Version. Auch bei MSX muß mit einer doppeltindizierten Variablen gearbeitet werden. Größter Unterschied besteht in der Bildschirmdarstellung.

Auf MSX wählen Sie die Auflösung von SCREEN 2, bei der die Grafikpunkte nur ein Viertel so groß sind wie beim Commodore- oder Atari-Programm. Entsprechend muß jeder Punkt vervierfacht werden, wie es z. B. schon beim Programm KALEIDOS.MSX gezeigt wurde. Oder Sie spielen auf einem in der Relation viermal so großen Platz.

Wenn Sie also ein MSX-System besitzen, dann sollten Sie jetzt einmal versuchen, die vorliegende Atari-Version für MSX-BASIC umzuschreiben.

4.4 Brettspiel

Gibt es ein traurigeres Bild, als einen einzelnen Menschen vor einem flimmernden Bildschirm, der mit einer oder, besser gesagt, gegen eine Maschine spielt? Wir leben in einer Zeit wachsender Vereinzelung und Einsamkeit. Die Computer springen in diese Lücke und bieten sich als Partner an. Aber eigentlich vertiefen sie nur unsere Isolation.

Dabei könnte das elektronische Medium genausogut *neue Ebenen menschlicher Begegnung* schaffen. Im Bereich Spiel könnten beliebig viele Mitspieler in beliebigen Entfernungen zusammentreffen. Der Computer kann Spielplan und Figuren darstellen und damit Spielformen ermöglichen, die in konventioneller Form nicht denkbar sind.

Das Spielfeld kann beliebige Ausdehnungen haben und sich in drei und mehr Dimensionen erstrecken. Es kann sich dynamisch verändern und verborgene Eigenschaften enthalten, die nur unter besonderen Voraussetzungen wirksam werden.

Die Spieler können gleichzeitig agieren. Zeit, Geschwindigkeit und Reaktionsvermögen kommen so ins Spiel und vermitteln eine weitere lebensnahe Komponente.

Das folgende Spiel soll einen kleinen Hinweis geben, was man mit dem elektronischen Spielbrett möglicherweise alles anstellen kann.

Auf dem Bildschirm erscheint ein quadratischer Plan, der schachbrettartig in zwölf mal zwölf Felder unterteilt ist. Jeder der beiden Spieler dirigiert eine Figur. In der Ausgangsposition befindet sich Spieler Blau in der oberen linken Ecke und Spieler Rot rechts unten.

Die Figuren werden über den *Joystick* gesteuert. Sie können senkrecht oder waagerecht jeweils ein Feld rücken. Das verlassene Feld verschwindet von der Bildfläche, so daß der Spielraum Schritt für Schritt enger wird. Der Spieler, der als erster kein freies Feld mehr findet, das er besetzen kann, hat verloren.

Soweit ließe sich dieses Spiel auch mehr oder weniger leicht mit konventionellem Spielmaterial ermöglichen. Das Programm ARENADUO bietet aber noch eine weitere Funktion, die so nur von einem Computer sinnvoll bereitgestellt werden kann.

Statt seine Figur zu ziehen, kann der Spieler auch durch Druck auf den Feuerknopf einen Sprung ins Ungewisse wagen. Das Programm ermittelt zufällig ein Spielfeld. Ist es frei, hat der Spieler Glück gehabt, und der Gegner kommt an die Reihe. Findet der Zufall jedoch ein bereits gelöschtes oder das vom Gegner besetzte Feld, so verschwindet die Figur ganz einfach und taucht nicht wieder auf: Das Spiel ist verloren.

Und so sieht das Listing dieses Strategie-Glücksspiels für Commodore aus:

```
0 REM ARENADUO.COM
10 T=1024:F=55296:R=RND(-TI)
20 PRINT CHR$(147):POKE 53280,12:POKE 53281,1
2
25 FOR X=8 TO 31:FOR Y=0 TO 23:POKE T+X+Y*40,
160:NEXT Y:NEXT X
30 FOR X=0 TO 22 STEP 2
40 C=C+1:IF C>1 THEN C=0
50 FOR Y=0 TO 22 STEP 2
60 C=C+1:IF C>1 THEN C=0
70 POKE F+8+X+Y*40,C:POKE F+9+X+Y*40,C:POKE F
+48+X+Y*40,C:POKE F+49+X+Y*40,C
80 NEXT Y
90 NEXT X
100 XA=0:YA=0:XZ=22:YZ=22
110 POKE F+8+XA+YA*40,6:POKE T+8+XA+YA*40,207
120 POKE F+9+XA+YA*40,6:POKE T+9+XA+YA*40,208
130 POKE F+48+XA+YA*40,6:POKE T+48+XA+YA*40,2
04
140 POKE F+49+XA+YA*40,6:POKE T+49+XA+YA*40,2
50
150 POKE F+8+XZ+YZ*40,2:POKE T+8+XZ+YZ*40,233
160 POKE F+9+XZ+YZ*40,2:POKE T+9+XZ+YZ*40,223
170 POKE F+48+XZ+YZ*40,2:POKE T+48+XZ+YZ*40,9
5
180 POKE F+49+XZ+YZ*40,2:POKE T+49+XZ+YZ*40,1
05
200 S=PEEK(56337):MA=0:NA=0:QQ=0:IF XA<0 THEN
 400
210 IF S<245 THEN GOTO 1000
220 IF S=255 THEN 200
230 IF S=254 THEN YA=YA-2:NA=2:QQ=1:IF YA<0 T
HEN YA=0:GOTO 200:REM YA=22
240 IF S=253 THEN YA=YA+2:NA=-2:QQ=1:IF YA>22
 THEN YA=22:GOTO 200:REM YA=0
250 IF S=251 THEN XA=XA-2:MA=2:QQ=1:IF XA<0 T
HEN XA=0:GOTO 200:REM XA=22
260 IF S=247 THEN XA=XA+2:MA=-2:QQ=1:IF XA>22
 THEN XA=22:GOTO 200:REM XA=0
270 IF PEEK(T+8+XA+YA*40)<>160 THEN XA=XA+MA:
YA=YA+NA:GOTO 200
280 IF QQ=0 THEN 200
290 POKE T+8+(XA+MA)+(YA+NA)*40,32:POKE T+8+X
A+YA*40,207:POKE F+8+XA+YA*40,6
300 POKE T+9+(XA+MA)+(YA+NA)*40,32:POKE T+9+X
A+YA*40,208:POKE F+9+XA+YA*40,6
310 POKE T+48+(XA+MA)+(YA+NA)*40,32:POKE T+48
+XA+YA*40,204:POKE F+48+XA+YA*40,6
320 POKE T+49+(XA+MA)+(YA+NA)*40,32:POKE T+49
+XA+YA*40,250:POKE F+49+XA+YA*40,6
330 FOR W=0 TO 150:NEXT W
400 S=PEEK(56336):MZ=0:NZ=0:QQ=0:IF XZ<0 THEN
 200
410 IF S<117 THEN GOTO 2000
```

```
420 IF S=127 THEN 400
430 IF S=126 THEN YZ=YZ-2:NZ=2:QQ=1:IF YZ<0 T
HEN YZ=0:GOTO 200:REM YZ=22
440 IF S=125 THEN YZ=YZ+2:NZ=-2:QQ=1:IF YZ>22
 THEN YZ=22:GOTO 200:REM YZ=0
450 IF S=123 THEN XZ=XZ-2:MZ=2:QQ=1:IF XZ<0 T
HEN XZ=0:GOTO 200:REM XZ=22
460 IF S=119 THEN XZ=XZ+2:MZ=-2:QQ=1:IF XZ>22
 THEN XZ=22:GOTO 200:REM XZ=0
470 IF PEEK(F+8+XZ+YZ*40)>1 THEN XZ=XZ+MZ:YZ=
YZ+NZ:GOTO 400
480 IF QQ=0 THEN 400
490 POKE T+8+(XZ+MZ)+(YZ+NZ)*40,32:POKE T+8+X
Z+YZ*40,233:POKE F+8+XZ+YZ*40,2
500 POKE T+9+(XZ+MZ)+(YZ+NZ)*40,32:POKE T+9+X
Z+YZ*40,223:POKE F+9+XZ+YZ*40,2
510 POKE T+48+(XZ+MZ)+(YZ+NZ)*40,32:POKE T+48
+XZ+YZ*40,95:POKE F+48+XZ+YZ*40,2
520 POKET+49+(XZ+MZ)+(YZ+NZ)*40,32:POKE T+49+
XZ+YZ*40,105:POKE F+49+XZ+YZ*40,2
530 FOR W=0 TO 150:NEXT W
600  GOTO 200
1000 REM UNTERPROGRAMM A-SPRUNG
1010 X1=INT(RND(1)*12)*2
1020 Y1=INT(RND(1)*12)*2
1030 POKE T+8+XA+YA*40,32:POKE T+9+XA+YA*40,3
2
1040 POKE T+48+XA+YA*40,32:POKE T+49+XA+YA*40
,32
1050 IF PEEK(T+8+X1+Y1*40)<>160 THEN XA=-3:YA
=-3:GOTO 400
1060 POKE T+8+X1+Y1*40,207:POKE F+8+X1+Y1*40,
6
1070 POKE T+9+X1+Y1*40,208:POKE F+9+X1+Y1*40,
6
1080 POKE T+48+X1+Y1*40,204:POKE F+48+X1+Y1*4
0,6
1090 POKE T+49+X1+Y1*40,250:POKE F+49+X1+Y1*4
0,6
1100 XA=X1:YA=Y1:GOTO 400
2000 REM UNTERPROGRAMM Z-SPRUNG
2010 X9=INT(RND(1)*12)*2
2020 Y9=INT(RND(1)*12)*2
2030 POKE T+8+XZ+YZ*40,32:POKE T+9+XZ+YZ*40,3
2
2040 POKE T+48+XZ+YZ*40,32:POKE T+49+XZ+YZ*40
,32
2050 IF PEEK(T+8+X9+Y9*40)<>160 THEN XZ=-3:YZ
=-3:GOTO 200
2060 POKE T+8+X9+Y9*40,233:POKE F+8+X9+Y9*40,
2
2070 POKE T+9+X9+Y9*40,223:POKE F+9+X9+Y9*40,
2
2080 POKE T+48+X9+Y9*40,95:POKE F+48+X9+Y9*40
,2
2090 POKE T+49+X9+Y9*40,105:POKE F+49+X9+Y9*4
0,2
2100 XZ=X9:YZ=Y9:GOTO 200
```

10: In den Variablen T und F werden die Basisadressen der beiden Bildschirmspeicherbereiche erfaßt. Mit Hilfe der Timer-Variablen TI wird ein Ausgangswert für die Zufallszahlenreihe gefunden.

20: Der Bildschirm wird gelöscht, Rand und Hintergrund bekommen die Farbe 12 (Grau).

25: Der Bildschirmbereich, in dem das Spielfeld dargestellt werden soll, wird mit vollen Leerzeichen (Zeichencode 160) beschrieben.

30: Jetzt können die Farbwerte für das Schachbrett im Speicher abgelegt werden. Das Spielfeld soll zwölf mal zwölf Felder groß werden. Jedes Spielfeld kann also zwei mal zwei Schreibstellen groß ausfallen. Der Schleifenzähler geht die Werte deshalb mit **STEP 2** durch.

40: Die Felder des Schachbrettes sind abwechselnd schwarz und weiß. Die Variable C bestimmt den Farbwert. Sie wechselt (in Zeile 60) zwischen 0 (Schwarz) und 1 (Weiß) hin und her. Da das letzte Feld der vorigen Spalte jedoch immer die gleiche Farbe hat wie das erste Feld der nächsten Spalte, ist an dieser Stelle ein zusätzlicher Farbwechsel notwendig.

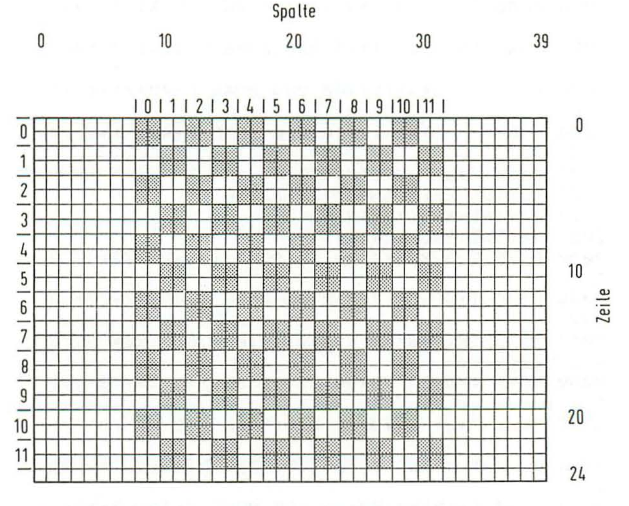


Abb. 11: Bei einem Schachbrett hat das letzte Feld der vorigen Spalte (oder Zeile) die gleiche Farbe wie das erste Feld der folgenden Spalte (oder Zeile)

50: Auch der Schleifenzähler der inneren FOR...NEXT-Schleife wird in Zweierschritten gezählt.

60: Der Farbwechsel zwischen Schwarz und Weiß wird erledigt.

70: Aufgrund der Koordinatenwerte X und Y werden die vier benachbarten Schreibstellen, die zusammen ein Spielfeld bilden, mit dem jeweiligen Farbwert gefüllt.

80: Ende der inneren Schleife.

90: Ende der äußeren Schleife.

100: Spieler Blau beginnt auf dem Spielfeld in der Ecke oben links. Die Variablen XA und YA halten seine Position. Rot steht zu Beginn unten links in der Ecke. Seine Position wird von XZ und YZ gehalten.

110 bis 140: Die vier benachbarten Schreibstellen, die das von Spieler Blau besetzte Spielfeld bilden, werden mit dem Farbwert 6 (Blau) und je einem Grafikzeichen beschrieben. Die vier Grafikzeichen sind so gewählt, daß sie zusammen die blaue Spielfigur bilden.

150 bis 180: Auf die gleiche Weise wird die rote Spielfigur durch vier andere Grafikzeichen dargestellt.

200: Der Steuerknüppel in Port A ist Spieler Blau zugeordnet. Hier wird sein Zustand gelesen. Die Variablen MA, NA und QQ werden auf 0 gesetzt. Wenn Blau bei einem Sprung ins Ungewisse im Leeren gelandet ist, dann hat XA einen negativen Wert bekommen. Ist XA kleiner als 0, dann wird die Kontrolle sofort an Spieler Rot übergeben; das Programm läuft bei Zeile 400 weiter; Blau kann nicht mehr ziehen.

210: Ist der Feuerknopf gedrückt, springt das Programm nach Zeile 1000, wo der Zufall ins Spiel kommt.

220: Wenn der Joystick in Ruhestellung steht, wird das Programm sofort zur erneuten Abfrage zurückgeführt. Rot kommt also erst an die Reihe, wenn Blau seinen Zug gemacht hat. Natürlich wäre es möglich, hier etwas anderes zu programmieren (siehe MSX-Version).

240 bis 270: Hier werden die verschiedenen Zustände des Steuerknüppels abgefragt und die entsprechenden Variablen XA oder YA geändert. In MA beziehungsweise NA wird der jeweils entgegen-

gesetzte Wert aufgehoben, um einen womöglich unzulässigen Zug rückgängig machen zu können. Wenn die Grenzwerte 0 beziehungsweise 22 erreicht sind, die Figur also am Rande des Spielfeldes steht, geht es in der Richtung nicht weiter. Es wäre aber auch denkbar, daß die Figur beim Überschreiten des Spielfeldrandes auf der genau gegenüberliegenden Seite wieder auftaucht. Damit würden sich ganz neue strategische Möglichkeiten anbieten, und ein anderer, nicht uninteressanter Reiz käme ins Spiel. Die entsprechend zu ändernden Werte sind als REM-Bemerkung angefügt. Probieren Sie es aus! Die Variable QQ überprüft, ob ein Zug erfolgt ist.

270: Die Figur darf nur rücken, wenn das Spielfeld leer ist. Um das zu überprüfen, reicht es aus, eine der vier Schreibstellen zu lesen, aus denen sich das Spielfeld zusammensetzt. Ist das Spielfeld bereits gelöscht, oder befindet sich die Figur des Gegners darauf, ist die Schreibstelle also nicht mit einer vollen beziehungsweise negativen Leerstelle gefüllt, werden die alten Werte von XA und YA mit Hilfe von MA und NA wiederhergestellt, und der Steuerknüppel wird erneut abgefragt.

280: Wenn kein zulässiger Zug erfolgt ist, hat QQ noch immer den Wert 0, und es wird ebenfalls zur Joystickabfrage zurückgesprungen. Diese Überprüfung ist notwendig, weil der Steuerknüppel z. B. in eine diagonale Position gedrückt werden kann, die von diesem Programm nicht verarbeitet wird.

290 bis 320: Wurde ein zulässiger Schritt gemacht, werden die betroffenen Felder entsprechend verändert. Die jeweils vier Schreibstellen des alten Feldes werden mit dem Leerzeichen (32) gefüllt und damit gelöscht. Der graue Hintergrund erscheint. Das neu betretene Feld wird mit dem Grafikzeichen der Spielfigur und dem Farbwert 6 (Blau) beschrieben. Die Figur ist ein Feld weitergerückt.

400 bis 520: Nun ist Spieler Rot an der Reihe. Dieser Programmblock ist mit dem vorherigen insoweit identisch, daß nur die Variablen entsprechend andere Namen haben.

1000: Hat Blau in Zeile 210 den Feuerknopf betätigt, dann gelangt das Programm in diese Zeile.

1010 und 1020 ermitteln zufällige Werte für die Position.

1030 und 1040: Die alte Position von Spieler Blau wird gelöscht, die Figur verschwindet.

1050: Wenn die zufällig ermittelte neue Position nicht frei ist, bekommen XA und YA negative Werte. Die Kontrolle wird durch Sprung nach Zeile 400 an Rot übergeben, und Blau ist aus dem Spiel.

1060 bis 1090: Wurde ein freies Feld gefunden, wird die Figur von Spieler Blau an dieser Stelle dargestellt.

1100: Die Koordinatenvariablen XA und YA bekommen die zufällig gewählten Werte X1 und Y1, und das Spiel wird mit dem Zug von Rot fortgesetzt.

2000 bis 2100 behandeln auf entsprechende Weise den Sprung von Rot.

Das Spiel ließe sich so programmieren, daß Zeit und Reaktion als aktive Spielfaktoren einbezogen würden. Zu diesem Zweck müßten die Sprunganweisungen so geändert werden, daß die Kontrolle nicht erst an den anderen Spieler übergeben wird, nachdem ein Zug gemacht wurde. Das MSX-Programm enthält die entsprechenden Änderungen der Sprungziele als REM-Hinweise.

```
0  REM ARENADUO.MSX
10 SCREEN 3:COLOR 15,14,14:CLS
20 DIM F(11,11)
30 FOR I=0 TO 11
40 C=C+1:IF C>1 THEN C=0
50 FOR J=0 TO 11
60 C=C+1:IF C>1 THEN C=0
70 F(I,J)=1+14*C
80 NEXT J
90 NEXT I
100 FOR I=0 TO 11
110 FOR J=0 TO 11
120 XO=32+I*16:YO=J*16
130 XU=47+I*16:YU=15+J*16
140 LINE (XO,YO)-(XU,YU),F(I,J),BF
150 NEXT J
160 NEXT I
170 XA=0:YA=0:XZ=11:YZ=11
180 F(0,0)=14:LINE (32,0)-(47,15),12,BF
```

```
190 F(11,11)=14:LINE (208,176)-(223,191),6,BF
200 S=STICK(1):MA=0:NA=0:QQ=0:IF XA<0 THEN 40
0
210 IF STRIG(1)=-1 THEN GOSUB 1000
220 IF S=0 THEN 200:REM 400
230 IF S=1 THEN YA=YA-1:NA=1:QQ=1:IF YA<0 THE
N YA=0:GOTO 200:REM löschen
240 IF S=5 THEN YA=YA+1:NA=-1:QQ=1:IF YA>11 T
HEN YA=11:GOTO 200:REM löschen
250 IF S=7 THEN XA=XA-1:MA=1:QQ=1:IF XA<0 THE
N XA=0:GOTO 200:REM löschen
260 IF S=3 THEN XA=XA+1:MA=-1:QQ=1:IF XA>11 T
HEN XA=11:GOTO 200:REM löschen
270 IF F(XA,YA)=14 THEN XA=XA+MA:YA=YA+NA:GOT
O 200:REM 400
280 IF QQ=0 THEN 200:REM löschen
290 LINE (32+XA*16,YA*16)-(47+XA*16,15+YA*16)
,12,BF:F(XA,YA)=14
300 X=XA+MA:Y=YA+NA:LINE (32+X*16,Y*16)-(47+X
*16,15+Y*16),14,BF
310 FOR W=0 TO 150:NEXT W:REM löschen
400 S=STICK(2):MZ=0:NZ=0:QQ=0:IF XZ<0 THEN 20
0
410 IF STRIG(2)=-1 THEN GOSUB 2000
420 IF S=0 THEN 400:REM 200
430 IF S=1 THEN YZ=YZ-1:NZ=1:QQ=1:IF YZ<0 THE
N YZ=0:GOTO 400:REM löschen
440 IF S=5 THEN YZ=YZ+1:NZ=-1:QQ=1:IF YZ>11 T
HEN YZ=1:GOTO 400;REM löschen
450 IF S=7 THEN XZ=XZ-1:MZ=1:QQ=1:IF XZ<0 THE
N XZ=0:GOTO 400:REM löschen
460 IF S=3 THEN XZ=XZ+1:MZ=-1:QQ=1:IF XZ>11 T
HEN XZ=11:GOTO 400:REM löschen
470 IF F(XZ,YZ)=14 THEN XZ=XZ+MZ:YZ=YZ+NZ:GOT
O 400:REM 200
480 IF QQ=0 THEN 400:REM löschen
490 LINE (32+XZ*16,YZ*16)-(47+XZ*16,15+YZ*16)
,6,BF:F(XZ,YZ)=14
500 X=XZ+MZ:Y=YZ+NZ:LINE (32+X*16,Y*16)-(47+X
*16,15+Y*16),14,BF
510 FOR W=0 TO 150:NEXT W:REM löschen
600 GOTO 200
1000 REM Unterprogramm A-Sprung
1010 X1=INT(RND(1)*12)
1020 Y1=INT(RND(1)*12)
1030 IF F(X1,Y1)=14 THEN LINE (32+XA*16,YA*16
)-(47+XA*16,15+YA*16),14,BF:XA=-3:YA=-3:RETUR
N 400
1040 LINE (32+X1*16,Y1*16)-(47+X1*16,15+Y1*16
),12,BF:F(X1,Y1)=14
1050 LINE (32+XA*16,YA*16)-(47+XA*16,15+YA*16
),14,BF:F(XA,YA)=14
1060 XA=X1:YA=Y1
1070 RETURN 400
2000 REM Unterprogramm Z-Sprung
2010 X9=INT(RND(1)*12)
2020 Y9=INT(RND(1)*12)
2030 IF F(X9,Y9)=14 THEN LINE (32+XZ*16,YZ*16
```

```
)-(47+XZ*16,15+YZ*16),14,BF:XZ=-3:YZ=-3:RETUR
N 200
2040 LINE (32+X9*16,Y9*16)-(47+X9*16,15+Y9*16
),6,BF:F(X9,Y9)=14
2050 LINE (32+XZ*16,YZ*16)-(47+XZ*16,15+YZ*16
),14,BF:F(XZ,YZ)=14
2060 XZ=X9:YZ=Y9
2070 RETURN 200
```

10: Wir wählen die Auflösung SCREEN 3; die Farbwerte werden bestimmt. Außerdem muß der Bildschirm gelöscht werden, weil ohne diesen Befehl die Änderung der Farbwerte vom System nicht ausgeführt wird.

20: Zur Erfassung des Spielstandes benötigen wir auch bei diesem Spiel eine zweite Ebene, die von einer doppeltindizierten Variablen gestellt wird.

30 bis 90: Der schachbrettartige Wechsel der Farbwerte wird, wie für das Commodore-Programm erklärt, besorgt. Da bei MSX Schwarz den Code 1 und Weiß den Code 15 hat, müssen die Farbwerte, entsprechend berechnet, in der Variablen F(n,n) erfaßt werden.

100 bis 160: Mit dem schon bekannten LINE-Befehl werden die Spielfelder gezeichnet; die doppeltindizierte Variable F bestimmt den Farbwert; der Parameter BF bewirkt, daß die beiden bestimmten Punkte als Ecken eines Kastens betrachtet werden und das so gezeichnete Rechteck mit der bestimmten Farbe ausgefüllt wird.

170: Die Anfangswerte für Spieler Blau und Rot.

180 und 190: Als Spielfigur bietet sich nur ein ganz mit der jeweiligen Farbe ausgefülltes Spielfeld an, das hier gezeichnet wird.

200 bis 300: Die Abfrage des Steuerknüppels und die entsprechenden Sprungbefehle oder Änderungen der Variableninhalte erfolgen ganz wie bei Commodore.

400 bis 500: Die Abfrage des Steuerknüppels von Spieler Rot geschieht genauso.

1000 bis 1070: Auch die Bearbeitung des Sprunges nach Betätigung des Feuerknopfes unterscheidet sich nur in den für MSX typischen Befehlen zur Darstellung von Grafik auf dem Bildschirm.

2000 bis 2070: Das gleiche gilt für den Sprung ins Ungewisse von Rot.

Da bei diesem Spiel fünf verschiedene Farbtöne benötigt werden, muß bei Atari eine sehr feine Auflösung für die Darstellung gewählt werden, weil nur GRAPHICS 10 so viele Farbtöne gleichzeitig auf der Mattscheibe zeigen kann.

```
0 REM ARENADUO.ATA
10 GRAPHICS 10:POKE 704,4:POKE 705,0:POKE 706
,14:POKE 707,134:POKE 708,38:POKE 709,42
20 FOR X=16 TO 60 STEP 4
30 C=C+1:IF C>2 THEN C=1
40 FOR Y=0 TO 188 STEP 16
50 C=C+1:IF C>2 THEN C=1
60 COLOR C:FOR I=0 TO 3:PLOT X+I,Y:DRAWTO X+I
,Y+15:NEXT I
70 NEXT Y
80 NEXT X
100 XA=0:YA=0:XZ=44:YZ=176
110 COLOR 0:FOR I=16 TO 19:PLOT XA+I,YA:DRAWT
O XA+I,YA+15:NEXT I
120 COLOR 3:FOR I=17 TO 18:PLOT XA+I,YA+1:DRA
WTO XA+I,YA+14:NEXT I
130 PLOT XA+16,YA+3:DRAWTO XA+19,YA+3:PLOT XA
+16,YA+13:DRAWTO XA+19,YA+13
140 COLOR 0:FOR I=16 TO 19:PLOT XZ+I,YZ:DRAWT
O XZ+I,YZ+15:NEXT I
150 COLOR 4:FOR I=17 TO 18:PLOT XZ+I,YZ+1:DRA
WTO XZ+I,YZ+14:NEXT I
160 PLOT XZ+16,YZ+6:DRAWTO XZ+19,YZ+6:PLOT XZ
+16,YZ+9:DRAWTO XZ+19,YZ+9
200 S=STICK(0):MA=0:NA=0:QQ=0:IF XA<0 THEN 40
0
210 IF STRIG(0)=0 THEN 1000
220 IF S=15 THEN 200
230 IF S=14 THEN YA=YA-16:NA=16:QQ=1:IF YA<0
THEN YA=0:GOTO 200
240 IF S=13 THEN YA=YA+16:NA=-16:QQ=1:IF YA>1
76 THEN YA=176:GOTO 200
250 IF S=11 THEN XA=XA-4:MA=4:QQ=1:IF XA<0 TH
EN XA=0:GOTO 200
260 IF S=7 THEN XA=XA+4:MA=-4:QQ=1:IF XA>44 T
HEN XA=44:GOTO 200
270 LOCATE XA+16,YA,L:IF L=0 THEN XA=XA+MA:YA
=YA+NA:GOTO 200
280 IF QQ=0 THEN 200
290 COLOR 0:FOR I=16 TO 19:PLOT XA+I,YA:DRAWT
O XA+I,YA+15:NEXT I
300 FOR I=16 TO 19:PLOT XA+MA+I,YA+NA:DRAWTO
XA+MA+I,YA+NA+15:NEXT I
310 COLOR 3:FOR I=17 TO 18:PLOT XA+I,YA+1:DRA
WTO XA+I,YA+14:NEXT I
320 PLOT XA+16,YA+3:DRAWTO XA+19,YA+3:PLOT XA
+16,YA+13:DRAWTO XA+19,YA+13
```

```
330 FOR W=0 TO 150:NEXT W
400 S=STICK(1):MZ=0:NZ=0:QQ=0:IF XZ<0 THEN 20
0
410 IF STRIG(1)=0 THEN 2000
420 IF S=15 THEN 400
430 IF S=14 THEN YZ=YZ-16:NZ=16:QQ=1:IF YZ<0
THEN YZ=0:GOTO 400
440 IF S=13 THEN YZ=YZ+16:NZ=-16:QQ=1:IF YZ>1
76 THEN YZ=176:GOTO 400
450 IF S=11 THEN XZ=XZ-4:MZ=4:QQ=1:IF XZ<0 TH
EN XZ=0:GOTO 400
460 IF S=7 THEN XZ=XZ+4:MZ=-4:QQ=1:IF XZ>44 T
HEN XZ=44:GOTO 400
470 LOCATE XZ+16,YZ,L:IF L=0 THEN XZ=XZ+MZ:YZ
=YZ+NZ:GOTO 400
480 IF QQ=0 THEN 400
490 COLOR 0:FOR I=16 TO 19:PLOT XZ+I,YZ:DRAWT
O XZ+I,YZ+15:NEXT I
500 FOR I=16 TO 19:PLOT XZ+MZ+I,YZ+NZ:DRAWTO
XZ+MZ+I,YZ+NZ+15:NEXT I
510 COLOR 4:FOR I=17 TO 18:PLOT XZ+I,YZ+1:DRA
WTO XZ+I,YZ+14:NEXT I
520 PLOT XZ+16,YZ+6:DRAWTO XZ+19,YZ+6:PLOT XZ
+16,YZ+9:DRAWTO XZ+19,YZ+9
530 FOR W=0 TO 150:NEXT W
600 GOTO 200
1000 REM UNTERPROGRAMM A-SPRUNG
1010 X1=INT(RND(0)*12)*4
1020 Y1=INT(RND(0)*12)*16
1030 COLOR 0:FOR I=16 TO 19:PLOT XA+I,YA:DRAW
TO XA+I,YA+15:NEXT I
1050 LOCATE X1+16,Y1,L:IF L=0 THEN XA=-3:YA=-
3:GOTO 400
1060 COLOR 0:FOR I=16 TO 19:PLOT X1+I,Y1:DRAW
TO X1+I,Y1+15:NEXT I
1070 FOR I=16 TO 19:PLOT X1+I,Y1:DRAWTO X1+I,
Y1+15:NEXT I
1080 COLOR 3:FOR I=17 TO 18:PLOT X1+I,Y1+1:DR
AWTO X1+I,Y1+14:NEXT I
1090 PLOT X1+16,Y1+3:DRAWTO X1+19,Y1+3:PLOT X
1+16,Y1+13:DRAWTO X1+19,Y1+13
1100 XA=X1:YA=Y1:GOTO 400
2000 REM UNTERPROGRAMM Z-SPRUNG
2010 X9=INT(RND(0)*12)*4
2020 Y9=INT(RND(0)*12)*16
2030 COLOR 0:FOR I=16 TO 19:PLOT XZ+I,YZ:DRAW
TO XZ+I,YZ+15:NEXT I
2050 LOCATE X9+16,Y9,L:IF L=0 THEN XZ=-3:YZ=-
3:GOTO 200
2060 COLOR 0:FOR I=16 TO 19:PLOT X9+I,Y9:DRAW
TO X9+I,Y9+15:NEXT I
2070 FOR I=16 TO 19:PLOT X9+I,Y9:DRAWTO X9+I,
Y9+15:NEXT I
2080 COLOR 4:FOR I=17 TO 18:PLOT X9+I,Y9+1:DR
AWTO X9+I,Y9+14:NEXT I
2090 PLOT X9+16,Y9+6:DRAWTO X9+19,Y9+6:PLOT X
9+16,Y9+9:DRAWTO X9+19,Y9+9
2100 XZ=X9:YZ=Y9:GOTO 200
```

10: Der Grafikmodus und die Farbwerte werden bestimmt.

20 bis 80: Das Schachbrettmuster wird auf die beschriebene Weise erzeugt.

100: Durch die gewählte hohe Auflösung ergeben sich auch entsprechend andere Startpositionen für Rot.

110: Das von der Figur besetzte Spielfeld wird mit Grau gefüllt, also schon gelöscht; dann wird in die Zeilen

120 und 130 die Figur hineingezeichnet. Die feine Auflösung wird genutzt, um eine entsprechend gestaltete Spielfigur zu erzeugen. Dadurch fällt das Programm etwas länger aus, ist aber immer noch deutlich kürzer als die Commodore-Version.

140 bis 160: Die entsprechenden Anweisungen für die anders gestaltete Figur von Rot.

200 bis 320: Die Abfrage des Steuerknüppels und die entsprechenden Anweisungen bleiben wie in den vorherigen Programmen. Nur die Befehle, die sich auf die Bildschirmdarstellung beziehen, fallen, den Bedingungen des Atari-BASIC entsprechend, anders aus.

400 bis 520: Das gleiche gilt für die Abfrage des Joysticks von Rot.

1000 bis 1100: Der Sprung ins Ungewisse von Spieler Blau . . .

2000 bis 2100: und der von Spieler Rot unterscheiden sich ebenfalls in den Grafikbefehlen.

5 Mehr Spiele

In diesem abschließenden Kapitel finden Sie vier aufwendigere Spieleprogramme, die entsprechend länger ausfallen. Diese Beispiele sollen Ihnen zeigen, was sich mit BASIC alles machen läßt, und sollen Sie zu eigenen Experimenten anregen.

Die kürzeren Beispiele in den vorhergehenden Abschnitten können den Kern abgeben für interessantere und ansehnlichere Programme. Sie können weitere Spielelemente hinzufügen , die Regeln verändern, die Grafik schöner gestalten und Toneffekte hinzufügen, die ganz besonders den Spielreiz steigern.

Jetzt wollen wir uns aber erst einmal die vier Spiele der folgenden Seiten ansehen.

5.1 Schlangentanz

Dieses Spiel knüpft an die Idee von ARENADUO an, ist allerdings als dynamisches Reaktionsspiel ausgelegt. Für die Spieler heißt das »schnell sein und aufpassen«.

Der Schlangenkampf wird auf einem Brett von 31 mal 22 Feldern ausgetragen. Die Spieler beginnen in den gegenüberliegenden Ecken als farbiger Grafikpunkt. Mit ihren Joysticks steuern sie über die Spielfläche und hinterlassen dabei eine neutrale gelbe Kriechspur. Erst wenn ein Spieler zweimal hintereinander ein Feld betritt, nimmt es seine Farbe an und kann hinfort vom Gegner nicht mehr betreten werden.

Natürlich geht es darum, möglichst viel Land zu gewinnen, während die Spielzeit als Grafikbalken am unteren Bildschirmrand

abläuft. Am Ende zählt der Rechner aus, wieviele Spielfelder jeder Spieler erobert hat.

Für den zusätzlichen Reiz ist das Spielgeschehen durchgehend mit Tönen unterlegt. Je nach Spieler und Bewegungsrichtung wird ein anderer Ton ausgegeben.

Das MSX-System verarbeitet die Sound-Befehle nicht direkt, sondern übergibt sie in einen *Puffer*, das ist ein Speicherbereich, der Daten sammelt und für den Abruf bereithält. In diesem Fall wird der Puffer vom *Tongenerator* gelesen, der parallel zum Prozessor, also von diesem unabhängig, arbeitet. Aber hören Sie selbst:

```
0   REM SNAKEDUO.MSX
10 GOSUB 1000
20 DIM F(30,21)
30 SCREEN 2:COLOR 12,14,14:CLS
40 FOR J=0 TO 21
50 FOR I=0 TO 30
60 C=C+1:IF C>1 THEN C=0
70 XO=8+I*8:YO=J*8
80 XU=15+I*8:YU=7+J*8
90 LINE(XO,YO)-(XU,YU),1+14*C,BF
100 NEXT I
110 NEXT J
120 X=30:Y=21
150 F(B,C)=2:LINE (8+B*8,C*8)-(15+B*8,7+C*8),
7,BF
160 F(X,Y)=-2:LINE (8+X*8,Y*8)-(15+X*8,7+Y*8)
,13,BF
170 H=H+1:IF H=DA*10 THEN P=P+8:OPEN"GRP:" FO
R OUTPUT AS #1:PSET(P,184),14:PRINT #1,CHR$(1
);CHR$(65):H=0:CLOSE #1
180 G=G+1:IF G=DA*310 THEN 420
190 B1=B:C1=C:A=STICK(1):Z=STICK(2)
210 IF A=0 THEN PLAY"O5C64":F(B,C)=2:GOTO 310
220 IF A=1 THEN C=C-1:PLAY"O5D64":IF C<0 THEN
 C=0
230 IF A=5 THEN C=C+1:PLAY"O5E64":IF C>21 THE
N C=21
240 IF A=7 THEN B=B-1:PLAY"O5F64":IF B<0 THEN
 B=0
250 IF A=3 THEN B=B+1:PLAY"O5G64":IF B>30 THE
N B=30
260 IF F(B,C)=-2 THEN PLAY"O5B64":B=B1:C=C1:G
OTO 310
270 IF F(B1,C1)>0 THEN F(B1,C1)=2:LINE (8+B1*
8,C1*8)-(15+B1*8,7+C1*8),4,BF
280 IF F(B1,C1)<1 THEN F(B1,C1)=1:LINE (8+B1*
8,C1*8)-(15+B1*8,7+C1*8),11,BF
290 LINE (8+B*8,C*8)-(15+B*8,7+C*8),7,BF
310 X1=X:Y1=Y
```

```
320 IF Z=0 THEN PLAY"O3C64":F(X,Y)=-2:GOTO 17
0
330 IF Z=1 THEN Y=Y-1:PLAY"O3D64":IF Y<0 THEN
 Y=0
340 IF Z=5 THEN Y=Y+1:PLAY"O3E64":IF Y>21 THE
N Y=21
350 IF Z=7 THEN X=X-1:PLAY"O3F64":IF X<0 THEN
 X=0
360 IF Z=3 THEN X=X+1:PLAY"O3G64":IF X>30 THE
N X=30
370 IF F(X,Y)=2 THEN PLAY"O3B64":X=X1:Y=Y1:GO
TO 170
380 IF F(X1,Y1)<0 THEN F(X1,Y1)=-2:LINE (8+X1
*8,Y1*8)-(15+X1*8,7+Y1*8),6,BF
390 IF F(X1,Y1)>-1 THEN F(X1,Y1)=-1:LINE (8+X
1*8,Y1*8)-(15+X1*8,7+Y1*8),11,BF
400 LINE (8+X*8,Y*8)-(15+X*8,7+Y*8),9,BF:GOTO
 170
420 SCREEN 0:COLOR 12,14,12:CLS:KEY OFF
430 LOCATE 18,6,0:PRINT"ENDE"
440 LOCATE 10,10:PRINT"Es folgt die Wertung"
450 FOR I=0 TO 30:FOR J=0 TO 21:S=S+F(I,J):NE
XT J:NEXT I
460 IF S>0 THEN COLOR 4,14,4:LOCATE 6,19:PRIN
T"Sieger: BLAU mit";ABS(S);"Punkten"
470 IF S<0 THEN COLOR 6,14,6:LOCATE 7,19:PRIN
T"Sieger: ROT mit";ABS(S);"Punkten"
480 IF S=0 THEN COLOR 1,14,1:LOCATE 7,14:PRIN
T"U N E N T S C H I E D E N"
490 LOCATE 5,20:PRINT"Neues Spiel? <J> Taste
drücken"
500 T$=INKEY$:IF T$="J" THEN RUN
510 GOTO 500
520 GOTO 510
1000 SCREEN 0:COLOR 12,1,8:CLS:KEY OFF:WIDTH
40
1010 LOCATE 0,3:PRINT"***********************
*****************";
1020 LOCATE 0,4:PRINT"*
                *";
1030 LOCATE 0,5:PRINT"* * * * *   S N A K E
D U O   * * * * *";
1040 LOCATE 0,6:PRINT"*
                *";
1050 LOCATE 0,7:PRINT"***********************
*****************";
1060 LOCATE 1,23:PRINT"(c) 1985 Karl-Heinz Ko
ch, Clausthal-Z."
1070 FOR W=0 TO 1000:NEXT W
1080 LOCATE 15,13:PRINT"Spieldauer"
1090 LOCATE 5,15:PRINT"Bitte geben Sie eine Z
ahl von"
1100 LOCATE 10,17:PRINT"drücken <RETURN>";:IN
PUT DA
1110 RETURN
```

10: Hier erfolgt ein Sprung in das Unterprogramm ab Zeile 1000, wo der Vorspann für das Spiel programmiert ist. Warum es vorteilhafter ist, einen solchen Programmblock in *unteren* BASIC-Zeilen abzulegen, erfahren Sie von den Programmiertricks des folgenden Kapitels.

20: Die doppeltindizierte Variable F(n,n) übernimmt die Schiedsrichterfunktion.

30: Für MSX nehmen wir wieder die Auflösung von SCREEN 2 und vervierfachen jeden Grafikpunkt, um auf die gewünschte Größe zu kommen. Sie können aber ebensogut einzelne Grafikpunkte als Spielfiguren verwenden und sich auf einem viermal so großen Spielfeld austoben. Dazu reichen wenige Änderungen in diesem Programm aus.

40 bis 110: Auf die schon erklärte Weise wird die Spielfläche mit einem Schachbrettmuster ausgefüllt. Da in der Zeile der Spielfläche eine ungerade Anzahl von Feldern liegt, erfolgt ein Farbwechsel vom letzten Feld der vorigen Zeile zum ersten der folgenden. Der bei ARENADUO notwendige zusätzliche Farbwechsel am Ende einer Zeile (oder Spalte) entfällt deshalb.

120: X und Y erhalten die Startkoordinaten für Spieler Rot. Blau beginnt auf 0,0. Da Variablen grundsätzlich den Anfangswert 0 haben, muß das für die Koordinatenvariablen B und C von Blau nicht extra programmiert werden.

150: Die blaue Spielfigur wird auf ihr Startfeld gesetzt;

160: ebenso die rote.

170: Hier wird die Spielzeit gemessen und ihr Verlauf grafisch dargestellt. Die Variable H zählt die Programmdurchläufe. Bei Spielbeginn muß eine Spieldauer eingegeben werden, die der Variablen DA zugeordnet wird.
Wenn H den zehnfachen Wert von DA erreicht hat, wird ein Datenkanal zum Grafikbildschirm geöffnet und ein Grafikzeichen ausgegeben. Der Befehl **PSET** bestimmt die Stelle, an der das Zeichen erscheint. Dabei wird die Spaltenposition durch die Variable P bestimmt, die vor jeder Ausgabe um den Wert 8 erhöht wird. Auf diese Weise erreicht man, daß die Grafikzeichen in einer Reihe nebeneinander erscheinen. H wird wieder auf 0 gesetzt, um erneut zum Zehnfachen von DA hochzuzählen.

180: Auch die Variable G zählt die Programmdurchläufe. Wenn G das 310fache von DA erreicht hat, ist die Spielzeit abgelaufen; und ab Zeile 420 wird das Resultat ermittelt.

190: Die Zustände von Steuerknüppel 1 und 2 werden in den Variablen A beziehungsweise Z erfaßt. Die Werte der Koordinatenvariablen B und C werden in B1 und C1 aufgehoben, um einen Zug rückgängig machen zu können, wenn das nötig werden sollte.

210: Wenn der Steuerknüppel von Blau in Ruhe ist, wird ein Ton gespielt, und zwar eine Vierundsechzigstelnote des C der fünften Oktave. In dem Element der doppeltindizierten Variablen F, das dem Spielfeld zugeordnet ist, auf dem sich Blau gerade befindet, wird eine 2 notiert. Dann wird die Kontrolle an Rot übergeben, ohne daß Blau einen Schritt gemacht hat!

Die Schiedsrichtervariable F(n,n) wird wie folgt verwendet: Der Grundwert aller Elemente ist 0. Betritt Blau ein Spielfeld zum ersten Mal, wird eine 1 notiert; betritt Blau ein Spielfeld, das bereits den Wert 1 hat, wird eine 2 geschrieben. Umgekehrt werden bei Rot −1 beziehungsweise −2 vermerkt. Felder mit dem Wert −1, 0 oder 1 dürfen von beiden Spielern betreten werden; Felder mit dem Wert −2 nur von Rot beziehungsweise mit 2 nur von Blau.

Wenn der Steuerknüppel in Ruhe ist, bekommt das Spielfeld den Wert 2 beziehungsweise −2, wird also vom betreffenden Spieler erobert. So haben die Spieler die Wahl, sich recht schnell fortzubewegen und nur Ansprüche auf Spielfelder anzumelden (Wert −1 beziehungsweise 1, Spielfeldfarbe Gelb) oder sich schrittweise, also langsam, zu bewegen, dafür aber die Spielfelder für sich zu erobern (Werte −2 beziehungsweise 2, Spielfeldfarben Rot beziehungsweise Blau).

220 bis 250 fragen die verschiedenen Richtungen ab, verändern entsprechend die Variablen B oder C, geben einen wechselnden Ton aus und überprüfen die Grenzwerte.

260: Wenn das angestrebte Feld mit dem Wert −2 belegt ist, also schon von Rot erobert wurde, wird ein Ton ausgegeben; die Koordinatenvariablen B und C erhalten ihre alten Werte B1 und C1 wieder, und die Kontrolle wird durch Sprung nach Zeile 310 an Rot übergeben. Blau hat keinen Zug gemacht.

270: Hat Blau einen zulässigen Zug gemacht, wird das alte verlassene Feld dunkelblau (4) gefärbt und bekommt den Wert 2, wenn es vorher bereits einen Wert größer als 0, also 1 oder 2, gehabt hat.

280: Hatte das verlassene Feld einen Wert kleiner als 1, also 0 oder −1, dann bekommt es jetzt den Wert 1 und wird in Farbe 11 (Gelb) getaucht.

290: Das neu betretene Feld wird in Hellblau (7) gefärbt, damit der Spieler immer eine Kontrolle hat, wo er sich auf dem Bildschirm befindet.

310 bis 400: Dieser Block verarbeitet die Bewegungen von Spieler Rot auf entsprechende Weise.

420: Wenn in Zeile 180 die Spielzeit verronnen ist, erfolgt die Wertung. Das Aktivieren von SCREEN 0 löscht die bestehende Grafik.

430 und 440: Textdarstellung.

450: Die Werte aller Spielfelder werden addiert und in der Variablen S zusammengefaßt.

460: Wenn S größer als 0 ist, also einen positiven Wert hat, dann waren mehr Felder von Blau (Wertung 1 oder 2) besetzt als von Rot (Wertung −1 oder −2). Also hat Blau um die Punktedifferenz gewonnen. Dieses Ergebnis wird hier ausgegeben.

470: Wenn S kleiner als 0 ist, dann hatte Rot mehr Felder belegt.

480: Nur wenn S genau den Wert 0 hat, dann hatten beide Spieler exakt gleich viele Felderpunkte. Das Spiel ist unentschieden.

490 bis 510: Durch Drücken der Taste J kann ein neues Spiel begonnen werden.

1000 bis 1070: Dieses Unterprogramm bringt den Titel auf die Mattscheibe.

1080 bis 1110: Zu Beginn müssen die Spieler eine Spieldauer eingeben.

In der Atari-Version sieht dieses Spiel ein wenig anders aus:

```
0 REM SNAKEDUO.ATA
10 REM EINE GR.1-ZEILE IN GR.0
20 GRAPHICS 0:POKE 559,0:POKE 712,14:POKE 752
,1
30 RESTORE 40:FOR I=0 TO 27:READ A:POKE 1536+
I,A:NEXT I
40 DATA 6,6,70,8,6,1,0,0,0,0,51,35,40,44,33,4
6,39,37,46,13,43,33,45,48,38,0,0,0,0,0
50 DL=PEEK(560)+PEEK(561)*256:POKE DL,1:POKE
DL+1,0:POKE DL+2,6
60 POKE 1542,PEEK(560)+3:POKE 1543,PEEK(561):
POKE 559,34
70 POSITION 7,23:? "(C) 1985 KARL-HEINZ KOCH"
:? :? :? :?
80 ? "Spieldauer? Bitte geben Sie eine Zahl v
on 1 (kurz) bis 20 (lang) ein und    druecken
 Sie <RETURN> ";
90 INPUT DAU
100 ? :? :? :? "Um das Spiel zu beginnen,
         bitte <START> druecken"
110 IF PEEK(53279)<>6 THEN 110
120 ? :? :? :? :? :? :? "          Bitte etwas
Geduld  ":? :?
130 DIM F(39,19):FOR M=0 TO 39:FOR N=0 TO 19:
F(M,N)=0:NEXT N:NEXT M:X=39:Y=19
140 GRAPHICS 3:POKE 708,198:POKE 709,52:POKE
710,88:POKE 712,90:POKE 752,1
150 COLOR 1:PLOT B,C:COLOR 2:PLOT X,Y:F(B,C)=
2:F(X,Y)=-2
160 FOR I=2 TO 37:? CHR$(160);:NEXT I:?
170 H=H+1:IF H=DAU*10 THEN ? CHR$(20);:H=0
180 G=G+1:IF G=DAU*360 THEN 420
190 SOUND 0,0,0,0:SOUND 2,0,0,0:B1=B:C1=C:A=S
TICK(0):Z=STICK(1)
205 REM STICK(0) ABFRAGEN
210 IF A=15 THEN SOUND 1,0,0,0:F(B,C)=2:GOTO
300
220 IF A=14 THEN C=C-1:SOUND 1,121,14,10:IF C
<0 THEN C=0
230 IF A=13 THEN C=C+1:SOUND 1,108,14,10:IF C
>19 THEN C=19
240 IF A=11 THEN B=B-1:SOUND 1,96,14,10:IF B<
0 THEN B=0
250 IF A=7 THEN B=B+1:SOUND 1,91,14,10:IF B>3
9 THEN B=39
255 REM STICK(0)-BEFEHLE UMSETZEN
260 IF F(B,C)=-2 THEN SOUND 0,60,6,15:B=B1:C=
C1:GOTO 300
270 IF F(B,C)=1 THEN COLOR 1:PLOT B,C:F(B,C)=
2:GOTO 300
280 IF F(B,C)=2 THEN COLOR 1:PLOT B,C:F(B,C)=
2:GOTO 300
290 COLOR 3:PLOT B,C:F(B,C)=1:COLOR 2:PLOT B,
C:COLOR 3:PLOT B,C:GOTO 310
300 COLOR 2:PLOT B,C:COLOR 1:PLOT B,C:COLOR 2
:PLOT B,C:COLOR 1:PLOT B,C:REM BLINKER
```

```
310 X1=X:Y1=Y
315 REM STICK(1) ABFRAGEN
320 IF Z=15 THEN SOUND 3,0,0,0:F(X,Y)=-2:GOTO
 410
330 IF Z=14 THEN Y=Y-1:SOUND 3,243,14,10:IF Y
<0 THEN Y=0
340 IF Z=13 THEN Y=Y+1:SOUND 3,217,14,10:IF Y
>19 THEN Y=19
350 IF Z=11 THEN X=X-1:SOUND 3,193,14,10:IF X
<0 THEN X=0
360 IF Z=7 THEN X=X+1:SOUND 3,182,14,10:IF X>
39 THEN X=39
365 REM STICK(1)-BEFEHLE UMSETZEN
370 IF F(X,Y)=2 THEN SOUND 2,243,6,15:X=X1:Y=
Y1:GOTO 410
380 IF F(X,Y)=-1 THEN COLOR 2:PLOT X,Y:F(X,Y)
=-2:GOTO 410
390 IF F(X,Y)=-2 THEN COLOR 2:PLOT X,Y:F(X,Y)
=-2:GOTO 410
400 COLOR 3:PLOT X,Y:F(X,Y)=-1:COLOR 1:PLOT X
,Y:COLOR 3:PLOT X,Y:GOTO 170
410 COLOR 1:PLOT X,Y:COLOR 2:PLOT X,Y:COLOR 1
:PLOT X,Y:COLOR 2:PLOT X,Y:GOTO 170:REM BLINK
ER
415 REM ENDE, AUSWERTUNG
420 FOR SO=0 TO 3:SOUND SO,0,0,0:NEXT SO:FOR
Z=0 TO 100:NEXT Z
430 ? "                    ENDE"
440 ? "          Es folgt die Wertung"
450 FOR M=0 TO 39:FOR N=0 TO 19:S=S+F(M,N):NE
XT N:NEXT M
460 IF S>0 THEN ? "Sieger GRUEN mit ";ABS(S);
" Punkten"
470 IF S<0 THEN ? "Sieger ROT mit ";ABS(S);"
Punkten"
480 IF S=0 THEN ? "            UNENTSCHIEDEN"
490 ? "Um ein neues Spiel zu beginnen, bitte
<OPTION> druecken."
500 IF PEEK(53279)=3 THEN RUN :REM ABFRAGEN D
ER OPTION-TASTE
510 GOTO 500
```

10 bis 60: Heimcomputer nutzen nur einen Teil der Bildfläche aus. Mit entsprechenden Hardware-Kenntnissen kann das Programm des Videoprozessors ANTIC, die sogenannte Display-Liste, geändert werden. Diese Zeilen enthalten eine kurze Maschinensprache-routine, die oberhalb des sonst vom Atari beschriebenen Bildschirmbereichs eine Textzeile in der Betriebsart GRAPHICS 1 darstellt.

70 bis 100: Textausgabe und Eingabe der Spieldauer.

110: Das Register wird abgefragt, in dem der Zustand der Funktionstasten gehalten wird.

120 und 130: Die Variable F(n,n) wird dimensioniert. Da eine dimensionierte numerische Variable bei Atari nicht grundsätzlich den Anfangswert 0 hat, beschreibt die doppelte FOR...NEXT-Schleife alle 800 Elemente. X und Y bekommen die Startkoordinaten für Spieler Rot. Das Atari-Spielfeld erstreckt sich über 40 mal 20 Felder.

140: Die Auflösung GRAPHICS 3 wird gewählt, Farbwerte werden definiert.

150: Die Spielfiguren Blau und Rot werden in ihre Ausgangspositionen gestellt, und die beiden Felder bekommen den Wert 2 beziehungsweise −2.

160: Ein Grafikbalken.

170: Die Darstellung der verrinnenden Spielzeit, durch einen wachsenden Balken von Grafikpunkten geleistet.

180: Überprüfung des Spielendes.

190: Mit dem Befehl **SOUND** werden bei Atari Töne und Geräusche ausgegeben. Die vier Parameter nach Sound bestimmen einen von vier möglichen Tonkanälen, die Tonhöhe, den Verzerrungsgrad und die Lautstärke. Ein gesetzter Ton bleibt auf dem bestimmten Tongenerator so lange stehen, bis ein anderer Ton angewiesen wird. Der Befehl **SOUND** 0,0,0,0 bringt den Tongenerator 0 zum Schweigen.

210 bis 250: Die Abfrage der Joystickzustände.

260 bis 290: Je nach Wert des betretenen Feldes wird die Anzeigefarbe bestimmt und der Wert des Feldes verändert.

300: Atari ermöglicht in dieser Auflösung nur vier Farben. Um die aktuelle Position des Spielers sichtbar zu machen, steht kein zusätzlicher Farbton zur Verfügung. Deshalb wird hier durch mehrfaches Setzen des gleichen Grafikpunktes in wechselnden Farben ein Blinkeffekt erzeugt.

320 bis 410: Der entsprechende Programmblock für Spieler Rot.

420 bis 510: Auswertung des Spielresultates.

5.2 Sumpfblüte

Dieses Spiel nutzt die Möglichkeit eines dynamischen Spielplans, der vom Rechner auf der Basis bestimmter Kontrollwerte fortlaufend verändert wird.

Das Spiel beginnt mit einer zufälligen Verteilung von grünen (Land) und blauen (Wasser) Grafikpunkten. Bei jedem Programmdurchlauf sucht sich der Rechner willkürlich einen Punkt auf dem Spielplan. Handelt es sich dabei um Wasser, so breitet er eine etwas größere Pfütze aus, findet er Land, legt er einen kleinen Bereich ein wenig trocken.

Diese Bedingungen für die Veränderung der Spiellandschaft könnten natürlich noch weiter verfeinert werden, indem der Computer in einem begrenzten Bereich ermittelt, welches Verhältnis von Wasser zu Land vorherrscht und in Abhängigkeit von diesem Wert verschieden dosierte Veränderungen vornimmt.

In der vorliegenden Fassung ist das Spiel für einen Spieler ausgelegt. Am Rande des Spielplans wandern zwei Zeiger auf der X- und auf der Y-Achse, wenn der Spieler den Steuerknüppel entsprechend bewegt. Durch Betätigen des Feuerknopfes schreibt der Spieler ein kleines quadratisches Stück Land dort auf den Spielplan, wo sich die von den beiden Zeigern angedeuteten Linien schneiden. Seine Aufgabe ist es, in der ablaufenden Zeit so viel Land wie möglich zu gewinnen.

Dieses Spiel ließe sich auch leicht für zwei Spieler umschreiben. Einmal wäre es möglich, daß jeder der beiden Spieler nur auf einen Zeiger einwirken kann; sie wären also Partner, die gemeinsam gegen die Natur ankämpfen, um den Sumpf trockenzulegen.

Die zweite Möglichkeit wäre, daß am linken und am oberen Rand zwei weitere Zeiger für den zweiten Spieler programmiert werden, der mit seinem Feuerknopf Wasserpfützen auf den Spielplan setzt. Es geht dann darum, welcher Spieler sich geschickter im Sumpf ausbreitet.

Da bei diesem Spiel mit einer recht hohen Auflösung gearbeitet wird, folgt nur eine Programmversion für Atari:

```
0 REM SUMPFEND.ATA
10 GOTO 1000
200 SOUND 0,2,4,8
210 COLOR 0:PLOT X-3,94:DRAWTO X,91:DRAWTO X+
3,94
220 PLOT X-2,95:DRAWTO X,93:DRAWTO X+2,95
230 COLOR 0:PLOT 158,Y-3:DRAWTO 155,Y:DRAWTO
158,Y+3
240 PLOT 159,Y-2:DRAWTO 157,Y:DRAWTO 159,Y+2
250 COLOR 3:PLOT X-3,95:DRAWTO X,92:DRAWTO X+
3,95
260 PLOT 159,Y-3:DRAWTO 156,Y:DRAWTO 159,Y+3
270 S=STICK(0)
280 IF S=15 THEN 340
290 X=X-((S=9) OR (S=10) OR (S=11)):IF X<6 TH
EN X=6
300 X=X+((S=5) OR (S=6) OR (S=7)):IF X>151 TH
EN X=151
310 Y=Y-((S=6) OR (S=10) OR (S=14)):IF Y<6 TH
EN Y=6
320 Y=Y+((S=5) OR (S=9) OR (S=13)):IF Y>87 TH
EN Y=87
330 SOUND 0,153,0,10
340 IF STRIG(0)=1 THEN 400
350 SCH=SCH+1:IF SCH=250 THEN POKE 710,144:GO
TO 530
360 COLOR 2:FOR V=-3 TO 3:FOR U=-3 TO 3
370 PLOT X+U,Y+V:NEXT U:NEXT V
380 SOUND 0,243,2,5
390 FOR Z=0 TO 50:NEXT Z
400 SOUND 0,0,0,0
410 ZYC=ZYC+1:IF ZYC=900 THEN POKE 710,144:GO
TO 530
420 XN=INT(RND(0)*146)+6
430 YN=INT(RND(0)*82)+6
440 LOCATE XN,YN,P
450 IF P=2 THEN 500
460 COLOR 1:PLOT XN-3,YN+1:DRAWTO XN-1,YN-3
470 DRAWTO XN+3,YN-1:DRAWTO XN+1,YN+3
480 DRAWTO XN-3,YN+1:PLOT XN-1,YN-1:DRAWTO XN
+1,YN-1
490 DRAWTO XN+1,YN+1:DRAWTO XN-1,YN+1
495 DRAWTO XN-1,YN-1:PLOT XN,YN:GOTO 520
500 COLOR 2:FOR U=-3 TO 3 STEP 2
510 FOR V=-3 TO 3 STEP 2:PLOT XN+U,YN+V:NEXT
V:NEXT U
520 GOTO 210
530 FOR G=3 TO 151 STEP 5
540 FOR H=3 TO 90 STEP 3
550 LOCATE G,H,V
560 IF V=1 THEN VE=VE+1
570 NEXT H
580 NEXT G
590 GRAPHICS 18
600 POSITION 3,1:? #6;"das ergebnis:"
610 V1=INT((VA/870)*100):V2=INT((VE/870)*100)
:VW=V1-V2
620 POSITION 0,3:? #6;"DER SUMPF HATTE EINE"
```

```
630 POSITION 0,4:? #6;"ANFANGS-FEUCHTIGKEIT"
640 POSITION 6,5:? #6;"VON ";V1;"%"
650 POSITION 3,7:? #6;"sie haben ";VW;"%"
660 POSITION 3,8:? #6;"trockengelegt"
670 POSITION 1,10:? #6;"FEUCHTIGKEIT JETZT":P
OSITION 8,11:? #6;V2;"%"
680 IF PEEK(53279)=6 THEN 200
690 GOTO 680
1000 GRAPHICS 18:POKE 710,146
1010 POSITION 2,2:? #6;"legen sie einen"
1020 POSITION 7,5:? #6;"SUMPF"
1030 POSITION 6,8:? #6;"trocken"
1040 FOR Z=0 TO 800:NEXT Z
1050 GRAPHICS 0:POKE 712,144:POKE 710,144:POK
E 709,4
1060 POKE 752,1:POKE 82,0
1070 ? "          Joystick in Port 0"
1080 ? "------------------------------------
---"
1090 ? "  Das Spiel beginnt mit einem Urknall
:"
1100 ? :? " Wasser und Land werden aus dem Ze
ntrum"
1110 ? :? "            herausgeschleudert."
1120 ? :? "        Legen Sie den Sumpf trocken!
"
1130 ? :? " Aber Sie haben nur begrenzte Zeit
 und"
1140 ? :? "nur einen bestimmten Vorrat an Dei
chen."
1150 ? :? "Fahren Sie mit dem Joystick hin un
d her"
1160 ? :? "  und deichen Sie mit dem Feuerkno
pf."
1170 ? :? :?
1175 ? "(c)1985 * Clausthal-Z. * Karl-Heinz K
och"
1180 ? "        Weiter: <START> tasten";
1190 IF PEEK(53279)=6 THEN 2000
1200 GOTO 1190
2000 GRAPHICS 23:POKE 712,144:POKE 708,130
2010 POKE 709,194:POKE 710,10:X=6:Y=6
2020 FOR U=3 TO 154:C=INT(RND(0)*2)+1:COLOR C
:PLOT U,3
2030 DRAWTO 157-U,90:SOUND 0,6-C,0,12-C*3:NEXT U
2040 FOR V=3 TO 90:C=INT(RND(0)*2)+1:COLOR C:
PLOT 154,V
2050 DRAWTO 3,93-V:SOUND 0,6-C,0,12-C*3:NEXT V
2060 COLOR 3:PLOT 2,95:DRAWTO 0,95:DRAWTO 0,0
2070 DRAWTO 159,0:DRAWTO 159,2:PLOT 159,10
2080 DRAWTO 159,95:DRAWTO 10,95
2100 FOR G=3 TO 151 STEP 5
2110 FOR H=3 TO 90 STEP 3
2120 LOCATE G,H,V
2130 IF V=1 THEN VA=VA+1
2140 NEXT H
2150 NEXT G
2200 GOTO 200
```

10: Im Unterprogramm ab Zeile 1000 werden Titel und Spielregeln auf den Bildschirm geschrieben. Dann breitet sich der Sumpf als malerisches Ereignis über die Bildfläche aus, und dazu ertönt das Rauschen eines Wasserfalls.

200: Ein kurzer Kontrollton.

210 bis 240: Mit COLOR 0 (Hintergrundfarbe) wird der Bereich rund um die alte Position des Zeigers gelöscht. In der Abbildung 12 sind diese Felder mit einem Punkt markiert.

250 und 260: Dann werden mit COLOR 3 (Weiß) die beiden Zeiger gezeichnet.

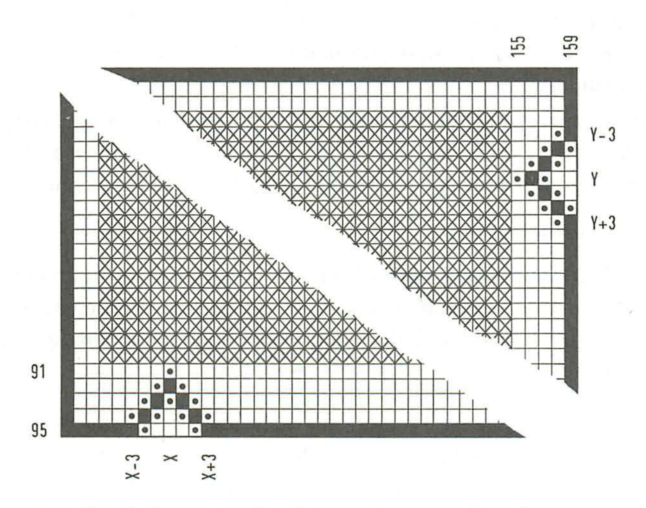


Abb. 12: Der Spielfeldbereich ist von einem weißen Rahmen umgeben, an dem entlang die beiden Zeiger laufen

270: Der Zustand des Steuerknüppels wird der Variablen S zugeordnet.

280: Wenn der Joystick in Ruhe ist (S=15), dann wird sofort zur Zeile 340 verzweigt, wo der Feuerknopf abgefragt wird.

290 bis 320: Hier werden die Zustände des Steuerknüppels umgesetzt. Bei diesem Spiel sind auch die vier diagonalen Positionen wirksam, es können also beide Zeiger gleichzeitig bewegt werden.

Natürlich könnte man jede der acht möglichen Stellungen des Joystick einzeln abfragen. In diesem Fall ist aber eine kürzere, allerdings auch anspruchsvollere Möglichkeit gewählt worden.
Es gibt drei Richtungen, die dazu führen, daß der Koordinatenwert X verringert werden muß; das sind unten links (S=9), links (S=11) und oben links (S=10). In dem Klammerausdruck erscheinen diese drei Werte durch den logischen Operator OR verknüpft. Wenn nur eine der durch OR verknüpften Bedingungen erfüllt ist, also wenn der Steuerknüppel in eine der drei genannten Richtungen bewegt wurde, bekommt die Klammer den Wert 1 (logischer Zustand: *wahr*) und die Zuordnung lautet X=X−1; X wird um 1 verringert, was einer Bewegung nach oben gleichkommt. Hat S keinen der drei vorgegebenen Werte, so ergibt der Ausdruck in der Klammer 0 (*falsch*); die Zuordnung lautet dann X=X−0; X bleibt dann also unverändert.

Auf entsprechende Weise werden die drei restlichen Richtungen überprüft, so daß insgesamt acht mögliche Bewegungen mit nur vier Programmzeilen abgefragt und umgesetzt werden können.

330: Ein Kontrollton.

340: Der Feuerknopf wird abgefragt. Wenn er nicht gedrückt ist (=1), wird das Programm bei Zeile 400 weiter bearbeitet. Sonst wird in den folgenden Zeilen ein grünes Quadrat auf dem Spielfeld trockengelegt.

350: Die Variable SCH zählt die Schüsse. Wenn der Wert 250 erreicht ist, hat der Benutzer sein Pulver verbraucht. Der Farbwert in Register 710 wird verändert. 144 ist der Farbwert des Hintergrundes, und Register 710 bestimmt den Farbwert für COLOR 3, die Farbe für Zeiger und Rahmen; kurz: Zeiger und Rahmen verschwinden. Danach wird ab Zeile 530 die Auswertung vorgenommen.

360 bis 390: Wenn der Schuß wirksam wird, erscheint an der augenblicklich von den beiden Zeigern markierten Position in Farbe 2 (Land) ein Quadrat von sieben mal sieben Grafikpunkten.

400: Tongenerator 0 wird ausgestellt.

410: Hier werden die Programmdurchläufe gezählt. Wenn der Spieler nicht vorher alle Schüsse verbraucht hat, endet das Spiel

nach 900 Durchläufen. Rahmen und Zeiger werden gelöscht. Dann wird zur Auswertung ab Zeile 530 verzweigt.

420 und 430: Jetzt ist der Rechner an der Reihe, den Spielplan zu verändern. Per Zufall wählt er die Koordinaten aus und

440 überprüft, welchen COLOR-Wert der Grafikpunkt an dieser Stelle hat.

450: Findet LOCATE den Wert 2 (Grün), geht es bei Zeile 500 weiter. Sonst wird in Zeile

460 bis 495 Farbe 1 (Blau) aktiviert und eine kleine Pfütze in den Sumpf gebettet. Danach läuft das Programm bei Zeile 520 weiter.

500 und 510: Hier wird die Farbe Grün eingeschaltet und um die zufällig gewählte Stelle herum eine lockere Verteilung von Land auf den Spielplan geschrieben.

520: Rücksprung nach Zeile 210 für den nächsten Durchlauf.

530 bis 580: Hier beginnt die Auswertung. Natürlich wäre es möglich, jeden einzelnen Grafikpunkt des Spielfeldes zu überprüfen, ob es sich um Wasser oder Land handelt. Allerdings besteht das Feld aus mehr als dreizehntausend Grafikpunkten! Diese Auszählung würde einfach zu lange Zeit benötigen.

Das Programm überprüft die Spielfläche deshalb in X-Richtung nur in Fünfer- und in Y-Richtung nur in Dreiersprüngen. Das Ergebnis ist zwar nur näherungsweise richtig, aber statistisch betrachtet doch ähnlich genug, um eine verwertbare Aussage zu ergeben.

590 bis 670: Das Ergebnis wird in % berechnet und mit großen, bunten Buchstaben (GRAPHICS 18) auf dem Bildschirm gezeigt.

680 und 690: Durch Drücken der START-Taste kann ein weiteres Spiel begonnen werden.

1000 bis 1180: Ausgabe des Titels und der Spielbeschreibung.

1190 bis 1200: Erst wenn der Benutzer START drückt, beginnt das Spiel. Er kann also den Text in Ruhe lesen und bestimmt selbst, wann es weitergehen soll.

2000 bis 2080: Die Sumpfgrafik wird unter dem Wohlklang von Wasserrauschen auf den Bildschirm gezaubert.

2100 bis 2200: Die anfängliche Wasser-/Land-Verteilung wird erhoben. Dann beginnt durch Sprung nach 200 das eigentliche Spiel.

5.3 Code-Knacker

Das Spielprinzip ist reichlich bekannt: Der Computer ermittelt zufällig eine Folge von fünf Ziffern. Der Benutzer soll diese fünfstellige Zahl erraten. Er gibt seinen Tip über die Tastatur ein und erfährt als Antwort, wie richtig oder falsch er geraten hat.

Für jede Ziffer, die der Spieler richtig eingegeben hat, erscheint ein schwarzer Punkt. Ist eine Ziffer nicht nur richtig, sondern steht sogar an der richtigen Stelle innerhalb der Zahl, dann erscheint ein weißer Punkt.

Die Aufgabe des Spielers besteht darin, aus möglichst wenigen Versuchen zu kombinieren, wie die richtige Zahl lautet. Er hat maximal zehn Versuche. Errät er die Zahl damit nicht, dann hat er das Spiel verloren, und der Rechner präsentiert ihm »schadenfroh« die richtige Zahl.

So einfach dieses Spiel im ersten Augenblick erscheinen mag – beim Programmieren ergeben sich einige Probleme. Betrachten wir zuerst die Commodore-Version:

```
0 REM CODEKNCK.COM
10 PRINT CHR$(147):T=1024:F=55296:POKE 53280,
8:POKE 53281,2
20 FOR J=0 TO 15:READ D:POKE F+132+J,1:POKE T
+132+J,D:NEXT J:RESTORE
30 DATA 42,32,3,15,4,5,45,11,14,1,3,11,5,18,3
2,42
40 PRINT:PRINT:PRINT:PRINT:PRINT:PRINT:PRINT:
PRINT"                    MASTER"
50 PRINT:PRINT:PRINT"                    YOUR"
60 PRINT:PRINT:PRINT"                    MIND"
70 PRINT:PRINT
72 PRINT:PRINT:PRINT:PRINT:PRINT:PRINT:PRINT"
     (C) 1985 KARL-HEINZ KOCH"
80 FOR W=0 TO 20:FOR J=0 TO 3:POKE F+458+J,10
90 FOR WW=0 TO 50: NEXT WW:POKE F+458+J,13:NE
XT J:NEXT W
100 PRINT CHR$(147):FOR J=0 TO 15:READ D:POKE
 F+52+J,1:POKE T+52+J,D:NEXT J
```

```
110 PRINT:PRINT:PRINT"   ERRATEN SIE DIE 5 GE
HEIMEN ZIFFERN"
120 PRINT:PRINT"      DES SICHERHEITS-COMPUT
ERS!"
130 PRINT:PRINT:PRINT"      GEBEN SIE  5 ZIFF
ERN MIT DEN"
140 PRINT:PRINT"         TASTEN <0> BIS <9> E
IN"
150 PRINT:PRINT:PRINT"EINE RICHTIG GERATENE Z
IFFER BEANTWORTET";
160 PRINT:PRINT"            DER COMPUTER MIT"
170 PRINT:PRINT"  WENN SIE AN DER RICHTIGEN S
TELLE STEHT";
180 PRINT:PRINT"  WENN SIE AN  DER FALSCHEN S
TELLE STEHT";
190 PRINT:PRINT:PRINT"ABER VORSICHT, SIE HABE
N NUR 10 VERSUCHE";
200 POKE F+680,1:POKE F+760,0:POKE T+680,81:P
OKE T+760,81
210 PRINT:PRINT"    WEITER: BELIEBIGE TASTE D
RUECKEN";
220 GET T$
230 X=INT(RND(1)*38)+1:Y=INT(RND(1)*21)+3:POK
E F+X+Y*40,0:IF T$="" THEN 220
240 POKE 53280,2
250 PRINT CHR$(147):RESTORE:FOR J=12 TO 27:RE
AD D:POKE F+J,1:POKE T+J,D:NEXT J
260 FOR I=0 TO 8
270 FOR J=3 TO 21:POKE F+9+I+J*40,14:POKE T+2
2+I+J*40,81:NEXT J
280 NEXT I
300 AC=INT(RND(1)*10)+48
310 BC=INT(RND(1)*10)+48
320 CC=INT(RND(1)*10)+48
330 DC=INT(RND(1)*10)+48
340 EC=INT(RND(1)*10)+48
350 ZZ=ZZ+2
360 GET T$:IF T$="" THEN 360
365 AU=ASC(T$):IF AU<48 OR AU>57 THEN 360
370 POKE T+9+(1+ZZ)*40,AU
380 GET T$:IF T$="" THEN 380
385 BU=ASC(T$):IF BU<48 OR BU>57 THEN 380
390 POKE T+11+(1+ZZ)*40,BU
400 GET T$:IF T$="" THEN 400
405 CU=ASC(T$):IF CU<48 OR CU>57 THEN 400
410 POKE T+13+(1+ZZ)*40,CU
420 GET T$:IF T$="" THEN 420
425 DU=ASC(T$):IF DU<48 OR DU>57 THEN 420
430 POKE T+15+(1+ZZ)*40,DU
440 GET T$:IF T$="" THEN 440
445 EU=ASC(T$):IF EU<48 OR EU>57 THEN 440
450 POKE T+17+(1+ZZ)*40,EU
460 AS=11:BS=11:CS=11:DS=11:ES=11
470 AX=AC:BX=BC:CX=CC:DX=DC:EX=EC
500 IF AU=AX THEN AS=1:AU=1:AX=0
510 IF BU=BX THEN BS=1:BU=1:BX=0
520 IF CU=CX THEN CS=1:CU=1:CX=0
530 IF DU=DX THEN DS=1:DU=1:DX=0
540 IF EU=EX THEN ES=1:EU=1:EX=0
```

```
550 IF AU=BX THEN AS=0:AU=1:BX=0
560 IF AU=CX THEN AS=0:AU=1:CX=0
570 IF AU=DX THEN AS=0:AU=1:DX=0
580 IF AU=EX THEN AS=0:AU=1:EX=0
590 IF BU=AX THEN BS=0:BU=1:AX=0
600 IF BU=CX THEN BS=0:BU=1:CX=0
610 IF BU=DX THEN BS=0:BU=1:DX=0
620 IF BU=EX THEN BS=0:BU=1:EX=0
630 IF CU=AX THEN CS=0:CU=1:AX=0
640 IF CU=BX THEN CS=0:CU=1:BX=0
650 IF CU=DX THEN CS=0:CU=1:DX=0
660 IF CU=EX THEN CS=0:CU=1:EX=0
670 IF DU=AX THEN DS=0:DU=1:AX=0
680 IF DU=BX THEN DS=0:DU=1:BX=0
690 IF DU=CX THEN DS=0:DU=1:CX=0
700 IF DU=EX THEN DS=0:DU=1:EX=0
710 IF EU=AX THEN ES=0:EU=1:AX=0
720 IF EU=BX THEN ES=0:EU=1:BX=0
730 IF EU=CX THEN ES=0:EU=1:CX=0
740 IF EU=DX THEN ES=0:EU=1:DX=0
750 Z=0
760 IF AS<BS THEN BH=BS:BS=AS:AS=BH:Z=Z+1
770 IF BS<CS THEN CH=CS:CS=BS:BS=CH:Z=Z+1
780 IF CS<DS THEN DH=DS:DS=CS:CS=DH:Z=Z+1
790 IF DS<ES THEN EH=ES:ES=DS:DS=EH:Z=Z+1
800 IF Z<>0 THEN 700
810 POKE F+22+(ZZ+1)*40,AS
820 POKE F+24+(ZZ+1)*40,BS
830 POKE F+26+(ZZ+1)*40,CS
840 POKE F+28+(ZZ+1)*40,DS
850 POKE F+30+(ZZ+1)*40,ES
860 EE=AS+BS+CS+DS+ES:IF EE=5 THEN 900
870 IF ZZ=20 THEN 950
880 GOTO 350
900 PRINT CHR$(147):PRINT:PRINT:PRINT:PRINT
910 PRINT"                 BRAVO!"
920 PRINT:PRINT"        SIE HABEN RICHTIG GERA
TEN!"
930 GOTO 980
950 PRINT CHR$(147):PRINT:PRINT:PRINT:PRINT
960 PRINT"         DIE ZAHL LAUTETE ";
970 PRINT CHR$(AC);"-";CHR$(BC);"-";CHR$(CC);
"-";CHR$(DC);"-";CHR$(EC)
980 PRINT:PRINT:PRINT:PRINT:PRINT"      NEUES
SPIEL?  BITTE <S> TASTEN"
990 GET T$:IF T$="S" THEN 250
1000 GOTO 990

READY.
```

10: Löschen des Bildschirms, Basisadressen der Bildschirmspeicher, Farbwerte für Rand und Hintergrund.

20: Der Befehl READ liest Daten aus BASIC-Zeilen, die mit **DATA** gekennzeichnet sind. READ beginnt bei der DATA-Zeile

mit der kleinsten Zeilennummer und liest, dort beginnend, fortlaufend die Daten aus dieser und den folgenden DATA-Zeilen. Dabei ist es für ein BASIC-Programm theoretisch ohne Bedeutung, welche Zeilennummern die DATA-Zeilen haben, ob sie also am Anfang des Programms stehen, an seinem Ende, oder beliebig verstreut sind. Dieses Programm enthält nur eine DATA-Zeile.
Bei jedem Durchlauf der FOR...NEXT-Schleife liest READ ein Datum und ordnet es der Variablen D zu. Ein Speicherbereich, der einem Teil der dritten Bildschirmzeile entspricht, bekommt den Farbwert 1 (Schwarz) und der entsprechende Speicher den Wert D. Der Befehl **RESTORE** bewirkt, daß der nächste READ-Befehl wieder von vorn beginnt, die DATA zu lesen.

30: Die DATA in dieser Zeile entsprechen den Bildschirmcodes für den Text »* CODE-KNACKER *«.

40 bis 72 schreiben den Titel auf den Bildschirm.

80 und 90 verändern laufend den Farbwert des Bildschirmbereichs, an dem das »YOUR« steht, so daß ein blitzender Farbeffekt erzeugt wird.

100: Der Bildschirm wird gelöscht, und der Spieltitel wird erneut geschrieben, dieses Mal allerdings in die erste Zeile.

110 bis 190 geben die Spielanleitung aus.

200: In den gePRINTeten Text werden der schwarze und der weiße Grafikpunkt mit POKE hineingesetzt.

210: Der Leser bestimmt, wann das Spiel beginnen kann.

220: GET liest die Tastatur.

230: Ein kleiner Gag: Solange der Benutzer keine Taste drückt, wird hier per Zufall eine Schreibstelle auf dem Bildschirm ermittelt und ihr Farbwert in Schwarz geändert. Die Schrift auf dem Bildschirm wird also, wie von einem Regenschauer getroffen, langsam schwarz.
Wenn T$ keinen Inhalt hat, wenn der Benutzer also noch keine Taste gedrückt hat, dann springt das Programm nach 220 zurück.

240: Die Randfarbe wird gewechselt.

250: Der Bildschirm wird gelöscht, der READ-Zeiger mit RESTORE wieder auf das erste Datum der ersten DATA-Zeile

gesetzt und der Spieletitel ein drittes Mal auf den Bildschirm geschrieben. Diesmal in die oberste, die nullte Zeile.

260 bis 280: Der Bildschirm wird für das Spiel vorbereitet. In die linke Hälfte werden Farbwerte geschrieben, so daß die vom Benutzer eingetippten Ziffern erscheinen können, in die rechte Hälfte Grafikzeichen, die farblich verändert das jeweilige Zwischenergebnis anzeigen. Die Aufteilung des Bildschirms zeigt die Abbildung 13.

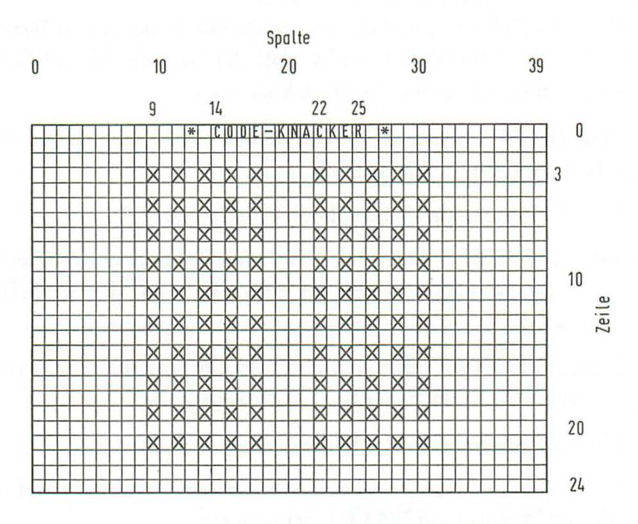


Abb. 13: Die Bildschirmaufteilung beim Spiel »Code-Knacker«. Nur die markierten Stellen werden beschrieben

300 bis 340: Der Computer ermittelt fünf Zufallszahlen zwischen 0 und 9. Die ASCII-Werte der Ziffern Null bis Neun sind 48 bis 57. Deshalb wird hier zu der gefundenen Zufallszahl der Wert 48 addiert. Die fünf Variablen AC, BC, CC, DC und EC halten die geheime, zu erratende Ziffernfolge.

350: Die Variable ZZ zählt die Anzahl der Rateversuche. Da mit ZZ auch die Zeile für die Bildschirmausgabe bestimmt, aber nur jede zweite Zeile der Mattscheibe beschrieben wird, zählt ZZ in Zweierschritten.

360: Die Tastatur wird mit GET abgefragt und das Programm so lange in dieser Zeile gehalten, bis eine Eingabe erfolgte.

365: Die eingegebene Taste wird durch die Funktion ASC in ihren ASCII-Wert umgewandelt und dieser Wert der Variablen AU zugeordnet. Dann wird überprüft, ob AU einen Wert von 48 bis 57 hat, also einer Ziffer von Null bis Neun entspricht. Hat AU irgendeinen anderen Wert, hat der Benutzer also keine Zifferntaste gedrückt, wird das Programm zur erneuten Abfrage nach 360 zurückgeführt.

370: Erfolgte eine zulässige Eingabe, wird unter Verwendung der Zählvariablen ZZ in die entsprechende Stelle des Bildschirmspeichers der Wert von AU geschrieben. Im Bereich der Ziffern sind der Commodore-Bildschirmcode und der ASCII-Code identisch.

380 bis 390: Auf die gleiche Weise wird die zweite vom Spieler getippte Ziffer in die Variable BU aufgenommen, überprüft und dargestellt.

400 bis 410: Ebenso die dritte Ziffer in Variable CU, ...

420 bis 430: die vierte in DU ...

440 bis 450: und die fünfte in EU. Damit hat der Spieler seinen Tip abgegeben, der auf dem Bildschirm gezeigt wird.

460: Die Variablen AS, BS, CS, DS und ES werden mit dem Wert 11 gefüllt. Diese Variablen werden das Ergebnis aufnehmen. Ihr Wert wird bestimmen, welche Farbe das Grafikzeichen 81 erhält. Farbwert 11 (Grau) gibt einen neutralen Wertungspunkt aus, der anzeigt, daß kein Rateergebnis erlangt wurde.

470: Da die Computerziffern, die in den Variablen AC bis EC gehalten werden, in den folgenden Operationen Veränderungen unterworfen werden müssen, wird ihr Wert hier auf die Variablen AX bis EX kopiert, um sie für weitere Ratedurchläufe zu erhalten.

500 bis 540: Der Rechner muß jetzt feststellen, wie die Eingabe des Spielers mit der von ihm ermittelten Zahl übereinstimmt. Ob zwei Ziffern an der gleichen Stelle identisch sind, ist recht leicht festzustellen. Es muß dann nämlich AU gleich AX, BU gleich BX usw. sein.
Wenn zwei Ziffern an gleicher Stelle gleich sind, dann bekommt die Wertungsvariable (AS bis ES) den Wert 1, der später einen weißen Punkt darstellen wird. Die beiden beteiligten Variablen werden

»entwertet«, damit sie bei den folgenden Vergleichen nicht noch einmal eine Wertung erbringen.

550 bis 740: Jetzt muß nämlich jede Ziffer des Spielers mit jeder Ziffer des Rechners verglichen werden, um festzustellen, ob Ziffern gleich sind, die nicht an der gleichen Stelle innerhalb der Zahl stehen.
Sind zwei gleiche Ziffern gefunden, wird die Wertungsvariable (AS bis ES) auf 0 gesetzt. Sie wird später dann einen schwarzen Punkt bewirken. Die beteiligten Variablen werden wieder »getilgt«, damit jede Ziffer auch wirklich nur einmal in die Wertung kommt.

750 bis 800: Jetzt haben wir das Ergebnis dieses Rateversuchs in den Variablen AS, BS, CS, DS und ES in Form von Farbwerten erfaßt. Die Variablen können aber nicht einfach in dieser Reihenfolge benutzt werden, denn daraus könnte der Spieler unerlaubte Rückschlüsse ziehen. Die Farbwerte müssen sortiert werden, so daß sie bei der Ausgabe geordnet auf dem Bildschirm erscheinen, und zwar erst die neutralen grauen Punkte, dann die schwarzen für richtige Ziffern und schließlich die weißen für richtige Ziffern an der richtigen Stelle.

Wie Variablen sortiert werden, haben wir ja bereits ganz am Anfang des Buches betrachtet. Hier ist uns dieses Wissen nun sehr nützlich. Da Commodore über keinen SWAP-Befehl verfügt, muß das Austauschen von Variablen über eine Hilfsvariable abgewickelt werden. Sie erinnern sich?

810 bis 850: Die sortierten Inhalte von AS bis ES werden mit Hilfe der Zählvariablen ZZ in die entsprechenden Zellen des Farbspeichers geschrieben.

860: Wenn eine richtige Ziffer an der richtigen Stelle getippt wurde, dann hat die entsprechende Variable AS bis ES den Wert 1. Wenn alle diese fünf Variablen den Wert 1 haben, dann sind alle Ziffern der Zahl richtig geraten; der Spieler hat gewonnen. Wenn also die Inhalte von AS bis ES addiert den Wert 5 ergeben, ist die Siegbedingung erfüllt. Das Programm springt nach Zeile 900.

870: Wenn die Zählvariable bei 20 angekommen ist (sie zählt in Zweierschritten), dann hat der Spieler seine zehn Versuche aufge-

braucht. Das Spiel ist ebenfalls beendet. Das Programm springt nach 950.

880: Wenn beide vorherigen Bedingungen nicht erfüllt waren, dann führt dieser Sprungbefehl zur nächsten Spielrunde nach Zeile 350.

900 bis 930: Das Siegergebnis wird ausgedruckt.

950 bis 970: Die Computerzahl wird aufgedeckt.

980 bis 1000: Der Spieler kann durch Drücken von Taste S ein neues Spiel beginnen.

In der Atari-Version dieses Programmes sieht hauptsächlich die Bildschirmdarstellung ganz anders aus:

```
0 REM CODEKNCK.ATA
10 GRAPHICS 2:POKE 708,244:POKE 709,182:POKE
710,0
11 POKE 711,244:POKE 712,0:POKE 752,1:POKE 82
,0:CC=14
12 ? "        (C)1985  Karl-Heinz Koch"
14 ? :? "          Weiter: <START> tasten";
16 POSITION 6,1:? #6;"MASTER"
18 POSITION 7,4:? #6;"your"
20 POSITION 7,7:? #6;"MIND"
22 CC=CC-2:IF CC<0 THEN CC=14
24 POKE 709,254-CC
26 IF PEEK(53279)<>6 THEN 22
28 GRAPHICS 0:POKE 710,192
29 POKE 709,4:POKE 752,1:POKE 82,0
30 ? :? " Erraten Sie die 5 geheimen Ziffern
des"
32 ? :? "          Sicherheit-Computerss"
34 ? :? :? "       Geben Sie 5 Ziffern mit den
"
36 ? :? "          Tasten <0> bis <9> ein"
38 ? :? :? "       Eine richtig geratene Ziffe
r"
40 ? :? " beantwortet der Computer mit einem
<=>"
42 ? :? " wenn sie an der richtigen Stelle st
eht"
44 ? :? "           sonst mit einem <->"
45 ? :? "*************************************
****";
46 ? "Aber Vorsicht! Sie haben nur 10 Versuch
e";
47 ? "****************************************
*";
48 ? :? "          Weiter: <START> tasten";
70 IF PEEK(53279)<>6 THEN 70
```

```
100 GRAPHICS 2:POKE 708,244:POKE 709,182:POKE
 710,0
105 POKE 711,52:POKE 712,0:POKE 752,1:ZZ=-1
110 AC=INT(RND(0)*10)+48
120 BC=INT(RND(0)*10)+48
130 CC=INT(RND(0)*10)+48
140 DC=INT(RND(0)*10)+48
150 EC=INT(RND(0)*10)+48
200 ZZ=ZZ+1
210 OPEN #1,4,0,"K:"
220 GET #1,AU:IF AU<48 OR AU>57 THEN 220
230 COLOR AU:PLOT 3,ZZ:SOUND 0,AU*3,10,ZZ+5
235 FOR Q=0 TO 100:NEXT Q:SOUND 0,0,0,0
240 GET #1,BU:IF BU<48 OR BU>57 THEN 240
250 COLOR BU:PLOT 4,ZZ:SOUND 0,BU*3,10,ZZ+5
255 FOR Q=0 TO 100:NEXT Q:SOUND 0,0,0,0
260 GET #1,CU:IF CU<48 OR CU>57 THEN 260
270 COLOR CU:PLOT 5,ZZ:SOUND 0,CU*3,10,ZZ+5
275 FOR Q=0 TO 100:NEXT Q:SOUND 0,0,0,0
280 GET #1,DU:IF DU<48 OR DU>57 THEN 280
290 COLOR DU:PLOT 6,ZZ:SOUND 0,DU*3,10,ZZ+5
295 FOR Q=0 TO 100:NEXT Q:SOUND 0,0,0,0
300 GET #1,EU:IF EU<48 OR EU>57 THEN 300
310 COLOR EU:PLOT 7,ZZ:SOUND 0,EU*3,10,ZZ+5
315 FOR Q=0 TO 100:NEXT Q:SOUND 0,0,0,0
350 CLOSE #1
360 AS=32:BS=32:CS=32:DS=32:ES=32
370 AX=AC:BX=BC:CX=CC:DX=DC:EX=EC
380 IF AU=AX THEN AS=61:AU=1:AX=0
390 IF BU=BX THEN BS=61:BU=1:BX=0
400 IF CU=CX THEN CS=61:CU=1:CX=0
410 IF DU=DX THEN DS=61:DU=1:DX=0
420 IF EU=EX THEN ES=61:EU=1:EX=0
430 IF AU=BX THEN AS=45:AU=1:BX=0
440 IF AU=CX THEN AS=45:AU=1:CX=0
450 IF AU=DX THEN AS=45:AU=1:DX=0
460 IF AU=EX THEN AS=45:AU=1:EX=0
470 IF BU=AX THEN BS=45:BU=1:AX=0
480 IF BU=CX THEN BS=45:BU=1:CX=0
490 IF BU=DX THEN BS=45:BU=1:DX=0
500 IF BU=EX THEN BS=45:BU=1:EX=0
510 IF CU=AX THEN CS=45:CU=1:AX=0
520 IF CU=BX THEN CS=45:CU=1:BX=0
530 IF CU=DX THEN CS=45:CU=1:DX=0
540 IF CU=EX THEN CS=45:CU=1:EX=0
550 IF DU=AX THEN DS=45:DU=1:AX=0
560 IF DU=BX THEN DS=45:DU=1:BX=0
570 IF DU=CX THEN DS=45:DU=1:CX=0
580 IF DU=EX THEN DS=45:DU=1:EX=0
590 IF EU=AX THEN ES=45:EU=1:AX=0
600 IF EU=BX THEN ES=45:EU=1:BX=0
610 IF EU=CX THEN ES=45:EU=1:CX=0
620 IF EU=DX THEN ES=45:EU=1:DX=0
700 Z=0
710 IF AS<BS THEN BH=BS:BS=AS:AS=BH:Z=Z+1
720 IF BS<CS THEN CH=CS:CS=BS:BS=CH:Z=Z+1
730 IF CS<DS THEN DH=DS:DS=CS:CS=DH:Z=Z+1
740 IF DS<ES THEN EH=ES:ES=DS:DS=EH:Z=Z+1
750 IF Z<>0 THEN 700
```

```
760 COLOR AS:PLOT 12,ZZ:GOSUB AS*100
770 COLOR BS:PLOT 13,ZZ:GOSUB BS*100
780 COLOR CS:PLOT 14,ZZ:GOSUB CS*100
790 COLOR DS:PLOT 15,ZZ:GOSUB DS*100
800 COLOR ES:PLOT 16,ZZ:GOSUB ES*100
810 EEE=AS+BS+CS+DS+ES:IF EEE=305 THEN 840
820 IF ZZ=9 THEN 900
830 GOTO 200
840 ? :? "  BRAVO! Sie haben richtig geraten.
":GOTO 910
900 ? :? "    Die Zahl lautete ";CHR$(AC);"-"
;
905 ? CHR$(BC);"-";CHR$(CC);"-";CHR$(DC);"-";
CHR$(EC);" !"
910 ? :? " Neues Spiel? Bitte <START> tasten!
";
920 IF PEEK(53279)=6 THEN GOTO 100
930 GOTO 920
3200 FOR X=0 TO 15 STEP 0.3:SOUND 0,32+7*X,14
,X:NEXT X:SOUND 0,0,0,0:RETURN
4500 FOR X=10 TO 0 STEP -0.2:SOUND 0,180,14,X
:NEXT X:SOUND 0,0,0,0:RETURN
6100 FOR X=-15 TO 15 STEP 0.5:SOUND 0,61,14,1
5-ABS(X):NEXT X:SOUND 0,0,0,0:RETURN
```

10 bis 20: Das Titelbild wird mit den großen, bunten Buchstaben von GRAPHICS 2 auf die Mattscheibe gebracht. In diesem Grafikmodus muß der PRINT-Befehl das Format PRINT #6; haben. #6 bezeichnet den Datenkanal zum Grafikbildschirm, der vom Benutzer nicht extra geöffnet (OPEN) werden muß.

22 und 24: Die Variable CC zählt von 14 bis 0 in Zweierschritten. Dann wird CC benutzt, um den Farbwert im Register 709 zu verändern. Atari kennt 256 Farbwerte, von denen aber bei den meisten Grafikbetriebsarten nur die geraden Werte verwendet werden können. Das Register 709 bestimmt den Farbton des Textes »YOUR«. Durch die fortlaufende Änderung des Farbwertes im Register entsteht ein blitzender Effekt.

26: Nur wenn der Benutzer START drückt, läuft das Programm weiter.

28 bis 48: Die Spielbeschreibung wird ausgegeben.

70: Nach Drücken von START beginnt das Spiel.

100 und 105: Der Spielverlauf wird auch in GRAPHICS 2 auf dem Bildschirm gezeigt. Statt farbiger Punkte wird das Rateergebnis mit den Zeichen = und – angezeigt.

110 bis 150: Der Rechner wählt seine Zahl.

200: Die Zählvariable ZZ zählt jeweils um 1 hoch. Der großen Buchstaben von GRAPHICS 2 wegen faßt der Bildschirm nur zwölf Zeilen.

210: Um beim Atari die Tastatur abzufragen, muß ein entsprechender Datenkanal geöffnet werden.

220 bis 315: Die fünf Ziffern des Spielers werden eingelesen, überprüft und den fünf Variablen zugeordnet. Zusätzlich wird bei jeder Eingabe ein Ton erzeugt, der abhängig von der getippten Ziffer ist. Das ergibt einen Sound-Effekt wie beim Wählen mit einem amerikanischen Tastentelefon.

350: Der Datenkanal muß auch wieder geschlossen werden.

360 bis 620: Jetzt müssen wieder alle Rateziffern mit allen Computerziffern verglichen werden. Im Verhältnis zur Commodore-Version ist hier nur anders, daß die Wertungsvariablen AS bis ES nicht Farbwerte 0, 1 oder 11 bekommen, sondern ASCII-Codes, und zwar 32 (Leerstelle) für nichts Erratenes, 61 (=) für eine richtige Ziffer an der richtigen Stelle oder 45 (–) für eine richtig geratene Ziffer irgendwo in der Zahl.

700 bis 750: Die Ergebnisvariablen werden sortiert.

760 bis 800: In GRAPHICS 2 kann mit dem COLOR-Befehl in Verbindung mit PLOT ein Zeichen auf den Bildschirm gebracht werden. Nach jeder Anzeige springt das Programm in ein Unterprogramm. Die Ergebnisvariable selbst bestimmt, in welches Unterprogramm gesprungen wird. In den drei verschiedenen Unterprogrammen werden Geräuschereignisse erzeugt. So wird das optische Ergebnis akustisch untermalt.

810: Die Siegbedingung wird überprüft.

820: Die Anzahl der Versuche wird überprüft.

830: Rücksprung nach 200, wenn das Spiel noch weiterläuft.

840: Das Siegergebnis wird angezeigt.

900 und 905: Der Computer verrät die richtige Zahl.

910 bis 930: Durch Drücken von START kann ein weiteres Spiel begonnen werden.

3200: Das Geräuschereignis für eine nicht erratene Stelle.

4500: Das Geräuschereignis für eine richtig erratene Ziffer.

6100: Das Geräuschereignis für eine richtige Ziffer an der richtigen Stelle.

5.4 Abenteuer

Eine Gattung von Computerspielen wird als Abenteuerspiele bezeichnet. Es geht im Prinzip immer darum, durch ein unbekanntes Gelände oder Gebäude einen Weg zu finden. Unterwegs lauern in der Regel Gefahren in Form von Fallen oder Feinden, und es sind irgendwelche Aufgaben zu lösen, Gegenstände zu finden oder sonstige Verbindungen herzustellen. Dadurch sind Abenteuerspiele nur mit großem Aufwand zu programmieren.

Bei anspruchsvolleren Spielen kommt noch hinzu, daß die Anweisungen an den Rechner in Form kurzer Sätze erfolgen wie »Säbel nehmen«, »Fenster öffnen« oder »Hexe fragen«. Dadurch werden die Programme noch um ein Weiteres komplizierter.

Wir wollen uns deshalb an dieser Stelle abschließend damit bescheiden, die Grundstruktur für ein Abenteuerspiel zu entwerfen und in ein spielbares Programm umzusetzen.

Der Spieler betritt die Eingangshalle eines verwunschenen Schlosses, wenn er den Fehler macht, dieses Spiel zu beginnen. Dieses Schloß hat zwölf Räume. Von jedem Raum aus kann sich der Abenteurer durch Betätigen der entsprechenden Tasten in die vier Himmelsrichtungen bewegen.

Allerdings ist das Schloß tatsächlich verwunschen, und man läuft leicht in die Irre. Das Ziel besteht ganz einfach darin, wieder ins Freie zu gelangen. Aber ob das wirklich so einfach ist, das sollten Sie ruhig einmal ausprobieren.

Versuchen wir es zuerst mit der MSX-Version:

```
0   REM ADVENTUR.MSX
100 SCREEN 1:COLOR 12,15,13
110 WIDTH 20:KEY OFF
120 LOCATE 0,3,0:PRINT"===================="
130 LOCATE 0,4:PRINT"=                  ="
```

```
140 LOCATE 0,5:PRINT"--A-D-V-E-N-T-U-R---"
150 LOCATE 0,6:PRINT"=                  ="
160 LOCATE 0,7:PRINT"===================="
170 LOCATE 0,12:PRINT"Ein Adventure-Spiel,"
180 LOCATE 2,14:PRINT"bei dem  Sie den"
190 LOCATE 1,16:PRINT"Ausgang aus  einem"
200 LOCATE 0,18:PRINT"verwunschenen Schloß"
210 LOCATE 3,20:PRINT"finden sollen."
220 LOCATE 0,23:PRINT"Weiter: <W> drücken.";
230 T$=INKEY$:IF T$="W" THEN 300
240 GOTO 230
300 SCREEN 1:COLOR 11,12,1
320 LOCATE 0,3,0:PRINT"====================="
330 LOCATE 0,4:PRINT"=                  ="
340 LOCATE 0,5:PRINT"--A-D-V-E-N-T-U-R---"
350 LOCATE 0,6:PRINT"=                  ="
360 LOCATE 0,7:PRINT"===================="
370 LOCATE 1,11:PRINT"Sie gehen von Raum"
380 LOCATE 0,12:PRINT"zu Raum.  Sie können"
390 LOCATE 1,13:PRINT"jedesmal vier ver-"
400 LOCATE 0,14:PRINT"schiedene Richtungen"
410 LOCATE 7,15:PRINT"wählen"
420 LOCATE 0,18:PRINT"Taste <N> für Norden"
430 LOCATE 0,19:PRINT"Taste <S> für  Süden"
440 LOCATE 0,20:PRINT"Taste <O> für  Osten"
450 LOCATE 0,21:PRINT"Taste <W> für Westen"
460 LOCATE 0,23:PRINT"Weiter: <W> drücken.";
470 T$=INKEY$:IF T$="W" THEN 1000
480 GOTO 470
1000 SCREEN 1:COLOR 1,15,14
1010 LOCATE 0,3:PRINT"********************"
1020 LOCATE 0,4:PRINT"********************"
1030 LOCATE 0,5:PRINT"**                **"
1040 LOCATE 0,6:PRINT"**  EINGANGSHALLE **"
1050 LOCATE 0,7:PRINT"**                **"
1060 LOCATE 0,8:PRINT"********************"
1070 LOCATE 0,9:PRINT"********************"
1100 LOCATE 0,18:PRINT"Taste <N> für Norden"
1110 LOCATE 0,19:PRINT"Taste <S> für  Süden"
1120 LOCATE 0,20:PRINT"Taste <O> für  Osten"
1130 LOCATE 0,21:PRINT"Taste <W> für Westen"
1200 T$=INKEY$
1210 IF T$="N" THEN 3000
1220 IF T$="S" THEN 10000
1230 IF T$="O" THEN 7000
1240 IF T$="W" THEN 4000
1250 GOTO 1200
2000 SCREEN 1:COLOR 1,14,1
2010 LOCATE 0,3:PRINT"********************"
2020 LOCATE 0,4:PRINT"********************"
2030 LOCATE 0,5:PRINT"**                **"
2040 LOCATE 0,6:PRINT"**    KORRIDOR    **"
2050 LOCATE 0,7:PRINT"**                **"
2060 LOCATE 0,8:PRINT"********************"
2070 LOCATE 0,9:PRINT"********************"
2100 LOCATE 0,18:PRINT"Taste <N> für Norden"
2110 LOCATE 0,19:PRINT"Taste <S> für  Süden"
2120 LOCATE 0,20:PRINT"Taste <O> für  Osten"
2130 LOCATE 0,21:PRINT"Taste <W> für Westen"
```

```
2200 T$=INKEY$
2210 IF T$="N" THEN 7000
2220 IF T$="S" THEN 1000
2230 IF T$="O" THEN 9000
2240 IF T$="W" THEN 3000
2250 GOTO 2200
3000 SCREEN 1:COLOR 1,4,6
3010 LOCATE 0,3:PRINT"********************"
3020 LOCATE 0,4:PRINT"********************"
3030 LOCATE 0,5:PRINT"**                **"
3040 LOCATE 0,6:PRINT"**  BANKETT-SAAL  **"
3050 LOCATE 0,7:PRINT"**                **"
3060 LOCATE 0,8:PRINT"********************"
3070 LOCATE 0,9:PRINT"********************"
3100 LOCATE 0,18:PRINT"Taste <N> für Norden"
3110 LOCATE 0,19:PRINT"Taste <S> für  Süden"
3120 LOCATE 0,20:PRINT"Taste <O> für  Osten"
3130 LOCATE 0,21:PRINT"Taste <W> für Westen"
3200 T$=INKEY$
3210 IF T$="N" THEN 6000
3220 IF T$="S" THEN 7000
3230 IF T$="O" THEN 1000
3240 IF T$="W" THEN 8000
3250 GOTO 3200
4000 SCREEN 1:COLOR 1,4,6
4010 LOCATE 0,3:PRINT"********************"
4020 LOCATE 0,4:PRINT"********************"
4030 LOCATE 0,5:PRINT"**                **"
4040 LOCATE 0,6:PRINT"**   BIBLIOTHEK   **"
4050 LOCATE 0,7:PRINT"**                **"
4060 LOCATE 0,8:PRINT"********************"
4070 LOCATE 0,9:PRINT"********************"
4100 LOCATE 0,18:PRINT"Taste <N> für Norden"
4110 LOCATE 0,19:PRINT"Taste <S> für  Süden"
4120 LOCATE 0,20:PRINT"Taste <O> für  Osten"
4130 LOCATE 0,21:PRINT"Taste <W> für Westen"
4200 T$=INKEY$
4210 IF T$="N" THEN 5000
4220 IF T$="S" THEN 8000
4230 IF T$="O" THEN 7000
4240 IF T$="W" THEN 6000
4250 GOTO 4200
5000 SCREEN 1:COLOR 11,6,12
5010 LOCATE 0,3:PRINT"********************"
5020 LOCATE 0,4:PRINT"********************"
5030 LOCATE 0,5:PRINT"**                **"
5040 LOCATE 0,6:PRINT"**   THRONSAAL    **"
5050 LOCATE 0,7:PRINT"**                **"
5060 LOCATE 0,8:PRINT"********************"
5070 LOCATE 0,9:PRINT"********************"
5100 LOCATE 0,18:PRINT"Taste <N> für Norden"
5110 LOCATE 0,19:PRINT"Taste <S> für  Süden"
5120 LOCATE 0,20:PRINT"Taste <O> für  Osten"
5130 LOCATE 0,21:PRINT"Taste <W> für Westen"
5200 T$=INKEY$
5210 IF T$="N" THEN 7000
5220 IF T$="S" THEN 2000
5230 IF T$="O" THEN 6000
5240 IF T$="W" THEN 10000
```

```
5250 GOTO 5200
6000 SCREEN 1:COLOR 8,14,1
6010 LOCATE 0,3:PRINT"********************"
6020 LOCATE 0,4:PRINT"********************"
6030 LOCATE 0,5:PRINT"**                **"
6040 LOCATE 0,6:PRINT"** RÜSTUNGSKAMMER **"
6050 LOCATE 0,7:PRINT"**                **"
6060 LOCATE 0,8:PRINT"********************"
6070 LOCATE 0,9:PRINT"********************"
6100 LOCATE 0,18:PRINT"Taste <N> für Norden"
6110 LOCATE 0,19:PRINT"Taste <S> für  Süden"
6120 LOCATE 0,20:PRINT"Taste <O> für  Osten"
6130 LOCATE 0,21:PRINT"Taste <W> für Westen"
6200 T$=INKEY$
6210 IF T$="N" THEN 2000
6220 IF T$="S" THEN 5000
6230 IF T$="O" THEN 7000
6240 IF T$="W" THEN 1000
6250 GOTO 6200
7000 SCREEN 1:COLOR 4,7,11
7010 LOCATE 0,3:PRINT"********************"
7020 LOCATE 0,4:PRINT"********************"
7030 LOCATE 0,5:PRINT"**                **"
7040 LOCATE 0,6:PRINT"**  PRIVATGEMACH  **"
7050 LOCATE 0,7:PRINT"**                **"
7060 LOCATE 0,8:PRINT"********************"
7070 LOCATE 0,9:PRINT"********************"
7100 LOCATE 0,18:PRINT"Taste <N> für Norden"
7110 LOCATE 0,19:PRINT"Taste <S> für  Süden"
7120 LOCATE 0,20:PRINT"Taste <O> für  Osten"
7130 LOCATE 0,21:PRINT"Taste <W> für Westen"
7200 T$=INKEY$
7210 IF T$="N" THEN 10000
7220 IF T$="S" THEN 6000
7230 IF T$="O" THEN 3000
7240 IF T$="W" THEN 8000
7250 GOTO 7200
8000 SCREEN 1:COLOR 4,3,12
8010 LOCATE 0,3:PRINT"********************"
8020 LOCATE 0,4:PRINT"********************"
8030 LOCATE 0,5:PRINT"**                **"
8040 LOCATE 0,6:PRINT"**   WACHSTUBE    **"
8050 LOCATE 0,7:PRINT"**                **"
8060 LOCATE 0,8:PRINT"********************"
8070 LOCATE 0,9:PRINT"********************"
8100 LOCATE 0,18:PRINT"Taste <N> für Norden"
8110 LOCATE 0,19:PRINT"Taste <S> für  Süden"
8120 LOCATE 0,20:PRINT"Taste <O> für  Osten"
8130 LOCATE 0,21:PRINT"Taste <W> für Westen"
8200 T$=INKEY$
8210 IF T$="N" THEN 4000
8220 IF T$="S" THEN 5000
8230 IF T$="O" THEN 2000
8240 IF T$="W" THEN 6000
8250 GOTO 8200
9000 SCREEN 1:COLOR 6,9,8
9010 LOCATE 0,3:PRINT"********************"
9020 LOCATE 0,4:PRINT"********************"
9030 LOCATE 0,5:PRINT"**                **"
```

```
9040 LOCATE 0,6:PRINT"**   WEINKELLER   **"
9050 LOCATE 0,7:PRINT"**                **"
9060 LOCATE 0,8:PRINT"********************"
9070 LOCATE 0,9:PRINT"********************"
9100 LOCATE 0,18:PRINT"Taste <N> für Norden"
9110 LOCATE 0,19:PRINT"Taste <S> für  Süden"
9120 LOCATE 0,20:PRINT"Taste <O> für  Osten"
9130 LOCATE 0,21:PRINT"Taste <W> für Westen"
9200 T$=INKEY$
9210 IF T$="N" THEN 12000
9220 IF T$="S" THEN 3000
9230 IF T$="O" THEN 10000
9240 IF T$="W" THEN 2000
9250 GOTO 9200
10000 SCREEN 1:COLOR 12,10,3
10010 LOCATE 0,3:PRINT"********************"
10020 LOCATE 0,4:PRINT"********************"
10030 LOCATE 0,5:PRINT"**                **"
10040 LOCATE 0,6:PRINT"**     KÜCHE      **"
10050 LOCATE 0,7:PRINT"**                **"
10060 LOCATE 0,8:PRINT"********************"
10070 LOCATE 0,9:PRINT"********************"
10100 LOCATE 0,18:PRINT"Taste <N> für Norden"
10110 LOCATE 0,19:PRINT"Taste <S> für  Süden"
10120 LOCATE 0,20:PRINT"Taste <O> für  Osten"
10130 LOCATE 0,21:PRINT"Taste <W> für Westen"
10200 T$=INKEY$
10210 IF T$="N" THEN 3000
10220 IF T$="S" THEN 5000
10230 IF T$="O" THEN 6000
10240 IF T$="W" THEN 8000
10250 GOTO 10200
11000 SCREEN 1:COLOR 1,6,8
11010 LOCATE 0,3:PRINT"********************"
11020 LOCATE 0,4:PRINT"********************"
11030 LOCATE 0,5:PRINT"**                **"
11040 LOCATE 0,6:PRINT"**  FOLTERKAMMER  **"
11050 LOCATE 0,7:PRINT"**                **"
11060 LOCATE 0,8:PRINT"********************"
11070 LOCATE 0,9:PRINT"********************"
11100 LOCATE 0,18:PRINT"Taste <N> für Norden"
11110 LOCATE 0,19:PRINT"Taste <S> für  Süden"
11120 LOCATE 0,20:PRINT"Taste <O> für  Osten"
11130 LOCATE 0,21:PRINT"Taste <W> für Westen"
11200 T$=INKEY$
11210 IF T$="N" THEN PLAY"O3CAB"
11220 IF T$="S" THEN :PLAY"O6CEDEGFGBAAFCEDEC
A":GOTO 13000
11230 IF T$="O" THEN PLAY"O4CAB"
11240 IF T$="W" THEN PLAY"O5CAB
11250 GOTO 11200
12000 SCREEN 1:COLOR 15,14,1
12010 LOCATE 0,3:PRINT"********************"
12020 LOCATE 0,4:PRINT"********************"
12030 LOCATE 0,5:PRINT"**                **"
12040 LOCATE 0,6:PRINT"**    VERLIES     **"
12050 LOCATE 0,7:PRINT"**                **"
12060 LOCATE 0,8:PRINT"********************"
12070 LOCATE 0,9:PRINT"********************"
```

```
12100 LOCATE 0,18:PRINT"Taste <N> für Norden"
12110 LOCATE 0,19:PRINT"Taste <S> für  Süden"
12120 LOCATE 0,20:PRINT"Taste <O> für  Osten"
12130 LOCATE 0,21:PRINT"Taste <W> für Westen"
12200 T$=INKEY$
12210 IF T$="N" THEN 10000
12220 IF T$="S" THEN 3000
12230 IF T$="O" THEN 11000
12240 IF T$="W" THEN 4000
12250 GOTO 12200
13000 SCREEN 1:COLOR 3,11,5
13010 LOCATE 0,3:PRINT"********************"
13020 LOCATE 0,4:PRINT"********************"
13030 LOCATE 0,5:PRINT"**                **"
13040 LOCATE 0,6:PRINT"**    FREIHEIT    **"
13050 LOCATE 0,7:PRINT"**                **"
13060 LOCATE 0,8:PRINT"********************"
13070 LOCATE 0,9:PRINT"********************"
```

100: Das gesamte Programm besteht aus einer größeren Menge von Bildschirminhalten, die alle im Textmodus SCREEN 1 gestaltet sind. Bei Textbetrieb können mit COLOR Farbwerte für Vordergrund, Hintergrund und Rand bestimmt werden.

110: Der Befehl **WIDTH** legt fest, wie viele Zeichen pro Zeile gezeigt werden; er bestimmt also den rechten und linken Rand des Bildschirms. KEY OFF unterdrückt die Anzeige der Funktionstasten am unteren Bildschirmrand.

120 bis 220 geben die Spielbeschreibung an den Bildschirm aus.

230 und 240: Der Benutzer bestimmt durch Drücken von Taste W, wann die nächste Bildschirmseite gezeigt wird.

300 bis 480: Der zweite Teil der Spielanleitung wird ausgegeben.

1000: Hier beginnt das eigentliche Spiel: Der mutige Abenteurer findet sich in der Eingangshalle wieder. Mit COLOR werden passende Farbwerte für die Anzeige bestimmt.

1010 bis 1070 zeigen an, in welchem Raum sich der Spieler gerade befindet.

1100 bis 1130 zeigen, welche Taste für welche Bewegung gedrückt werden muß.

1200: Die Tastatur wird gelesen.

1210 bis 1240: Je nach Tasteneingabe wird das Programm zu einer anderen Zeile verzweigt, und das bedeutet, zu einem anderen

Raum. Des Rätsels Lösung für dieses Verwirrspiel versteckt sich also in den Sprunganweisungen.

1250: Erfolgte keine oder keine zulässige Tastatureingabe, dann bleibt das Programm in diesem Raum, und die Tastatur wird erneut abgefragt. Computer sind ja so geduldig.

2000: Durch den SCREEN-Befehl wird ein alter Bildschirminhalt gelöscht. COLOR bestimmt für den Raum passende Farbwerte, die auch ein wenig Stimmung vermitteln und eine optische Orientierung ermöglichen sollen.

2010 bis 2070: Der Korridor.

2100 bis 2130: Der Wegweiser.

2200 bis 2250: Tastaturabfrage mit entsprechenden Sprunganweisungen.

3000 bis 3250: Der Bankett-Saal.

4000 bis 4250: Die Bibliothek.

5000 bis 5250: Der Thronsaal.

6000 bis 6250: Die Rüstungskammer.

7000 bis 7250: Ein Privatgemach.

8000 bis 8250: Die Wachstube.

9000 bis 9250: Der Weinkeller.

10000 bis 10250: Die Küche.

11000 bis 11250: Die Folterkammer.

12000 bis 12250: Das Verlies.

13000 bis 13070: Wenn Sie es tatsächlich geschafft haben, einen Weg hierher zu finden, dann haben Sie zwar die Lösung gefunden. Aber wenn Sie nicht mitgeschrieben haben, welchen Weg Sie gegangen sind, dann werden Sie beim zweiten Versuch wieder genauso viele Schwierigkeiten haben, aus dem verwunschenen Schloß hinauszufinden.
Wenn Sie Spaß daran haben, können Sie dieses Abenteuergerüst beliebig ausbauen, in den Räumen Gegenstände oder Rätsel verstecken und das Erreichen bestimmter wichtiger Räume vom Lösen dieser Aufgaben abhängig machen.

In der Commodore-Version gibt es die Möglichkeit, Textbildschirme in allen sechzehn Farben zu gestalten, die wir auch unbedingt ausnutzen wollen.

Aus Platzgründen wird nicht das gesamte Programm abgedruckt. Da die Programmblöcke für die einzelnen Räume quasi identisch sind, zeigt das folgende Listing nur die BASIC-Zeilen, um den Bildschirminhalt für einen einzigen Raum auf die Mattscheibe zu bekommen. Sie können sich an der MSX-Version orientieren, wenn Sie nach diesem Muster die Anweisungen für die restlichen elf Räume programmieren:

```
0 REM ADVENTUR.COM
10 T=1024:F=55296
1000 PRINT CHR$(147)
1010 POKE 53280,0:REM RANDFARBE
1020 POKE 53281,11:REM HINTERGRUNDFARBE
1030 FOR J=2 TO 8
1040 FOR I=10 TO 29
1050 POKE T+I+J*40,42:REM SCHMUCKRAND AUS "*"
1060 POKE F+I+J*40,7:REM FARBE SCHMUCKRAND
1070 NEXT I
1080 NEXT J
1090 FOR J=4 TO 6
1100 FOR I=12 TO 27
1110 POKE T+I+J*40,32:REM TEXTFLAECHE LOESCHE
N
1120 POKE F+I+J*40,8:REM FARBE TITELTEXT
1130 NEXT I
1140 NEXT J
1150 FOR I=12 TO 27:READ D:POKE T+I+200,D:NEX
T I:REM TITELZEILE
1160 DATA 32,32,5,9,14,7,1,14,7,19,8,1,12,12,
5,32:REM BILDSCHIRMCODE:TITEL
1200 FOR I=10 TO 29:POKE F+I+640,14:NEXT I:RE
M 1. HINWEISZEILE BLAU
1210 FOR I=10 TO 29:POKE F+I+720,8:NEXT I:REM
 2.ZEILE ORANGE
1220 FOR I=10 TO 29:POKE F+I+800,7:NEXT I:REM
 3. ZEILE GELB
1230 FOR I=10 TO 29:POKE F+I+880,5:NEXT I:REM
 4. ZEILE GRUEN
1240 FOR J=0 TO 6 STEP 2
1250 FOR I=10 TO 29
1260 READ D:POKE T+I+640+J*40,D
1270 NEXT I
1280 NEXT J
1290 DATA 20,1,19,20,5,32,60,14,62,32,45,45,6
2,32,14,15,18,4,5,14:REM 1. ZEILE
1300 DATA 20,1,19,20,5,32,60,19,62,32,45,45,6
2,32,19,21,5,4,5,14:REM 2. ZEILE
1310 DATA 20,1,19,20,5,32,60,15,62,32,45,45,6
2,32,32,15,19,20,5,14:REM 3. ZEILE
```

```
1320 DATA 20,1,19,20,5,32,60,23,62,32,45,45,6
2,32,23,5,19,20,5,14:REM 3. ZEILE
1330 GET T$
1340 IF T$="N" THEN 3000
1350 IF T$="S" THEN 10000
1360 IF T$="O" THEN 7000
1370 IF T$="W" THEN 4000
1380 GOTO 1330
```

10: Die Basisadressen der Bildschirmspeicher werden wieder in Variablen gefaßt.

1000: Der Bildschirm(speicher) muß gelöscht werden. Dann wird...

1010: der Farbwert für den Rand und...

1020: der für die Hintergrundfarbe bestimmt.

1030 bis 1080: An die gewünschte Stelle des Bildschirms wird ein Schmuckrand aus Sternchen (Zeichencode 42) geschrieben. Die Farbe für den Schmuckrand kann für jeden Raum anders bestimmt werden.

1100 bis 1140: Weil es einfacher zu programmieren ist, wurde oben eine ganze Fläche von Sternchen geschrieben. Jetzt wird die Innenfläche gelöscht, so daß tatsächlich ein Rand entsteht. Außerdem wird für die Innenfläche ein anderer Farbwert bestimmt.

1150: In diese Innenfläche wird nun ein Text, die Bezeichnung des Raumes, geschrieben. Dazu müssen die Codes der einzelnen Buchstaben in die entsprechenden Speicherzellen des Zeichenspeichers geschrieben werden. Die FOR...NEXT-Schleife läuft von 12 bis 27 und bestimmt damit die Spaltenpositionen für den Text. Mit READ werden die Zeichencodes aus der DATA-Zeile gelesen. Da der Titel »EINGANGSHALLE« nicht die ganze Breite ausfüllt, beginnen die DATA mit 32 (Leerzeichen).

1200 bis 1230: Jede dieser vier Zeilen schreibt einen anderen Farbwert in einen anderen Speicherbereich. Dadurch bekommen weiter unten die Hinweistexte wechselnde Farben, die dem Betrachter anzeigen, welche Taste er für welche Bewegungsrichtung drücken muß.

1240 bis 1280 schreiben die Zeichencodes für diese vier Hinweiszeilen in die entsprechenden Speicherbereiche.

1290 bis 1320 enthalten die Zeichencodes für den darzustellenden Text. Die Aufteilung des Bildschirms zeigt Ihnen die Abbildung 14.

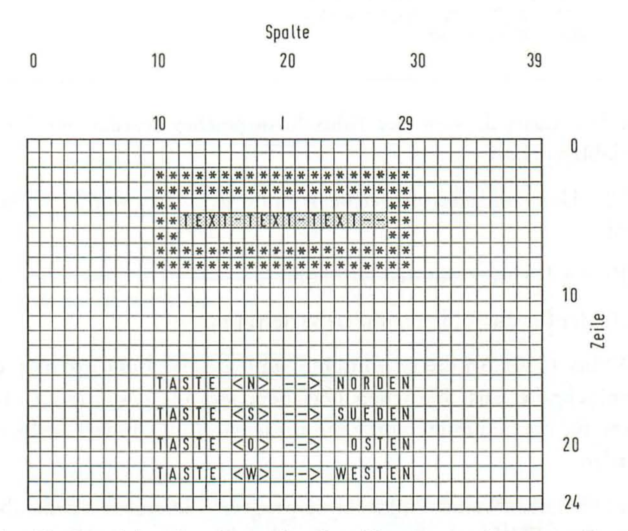


Abb. 14: So sieht eine Textseite des Abenteuerspiels aus. Es verbleibt viel Platz für individuelle Änderungen

1330: GET liest die Tastatur.

1340 bis 1370: Je nach eingegebener Taste wird ein anderer Sprung zu einem anderen Programmblock, also in ein anderes Zimmer des verwunschenen Schlosses, ausgeführt.

1380: Erfolgte keine Eingabe, die das Programm verarbeitetn kann, so wird die Tastatur erneut abgefragt.

6 Programmierhilfen

In diesem Kapitel möchte ich Ihnen noch ein paar Tips und Tricks verraten, die Ihnen das Schreiben, aber auch das Abtippen von Programmen erleichtern können. Oft sind es nur Kleinigkeiten, die viel Arbeit ersparen, wenn man um sie weiß.

Im ersten Abschnitt will ich Ihnen zeigen, wie man Programme umschreibt, korrigiert oder sonstwie verändert. Man spricht vom Editieren (oder Edieren).

Jedes Programm, das Sie schreiben oder abtippen, wird ein paar Fehler enthalten. Wie Sie die Störenfriede schneller finden und ausmerzen, soll Ihnen der zweite Abschnitt zeigen.

Um aus BASIC-Programmen das Letzte herauszuholen, muß man schon ein wenig über die internen Vorgänge des Rechners wissen. Im dritten Abschnitt will ich Ihnen zeigen, worauf Sie achten müssen, um ein möglichst wirkungsvolles Programm zu schreiben.

6.1 Editieren

Vor die Segnungen des Heimcomputers haben die Götter die Mühe der Programmeingabe gesetzt. Manche unnütze Arbeit kann man sich allerdings ersparen, wenn man weiß, wie's gemacht wird.

Wenn Sie Ihren Rechner einschalten, steht oben links in der Ecke ein weißes Quadrat, der *Cursor* (Läufer). Er zeigt an, wo das nächste Zeichen dargestellt wird, das Sie eintippen, und hilft so bei der Orientierung auf dem Bildschirm.

Der Cursor wird über vier *Cursortasten* (oft nur zwei und **SHIFT**) nach links, rechts oben und unten gesteuert. Außerdem gibt es noch eine Taste, meist mit **HOME** beschriftet, die den Cursor

zurück in die linke obere Ecke stellt. Bei gleichzeitigem Drücken von SHIFT wird dabei der gesamte Bildschirm gelöscht (**CLEAR HOME**).

Bei manchen Systemen kann sich der Cursor auch *verändern* und dadurch den Betriebszustand (z. B. Sinclair) oder die Funktion »Einfügen« (**INSERT**) (z. B. MSX) anzeigen.

BASIC-Zeilen beginnen immer mit einer *Zeilennummer*. Die Zeilennummern bestimmen, in welcher Reihenfolge die einzelnen Anweisungen später durch den Rechner bearbeitet werden. Üblicherweise werden die Zeilen in Zehnerschritten numeriert, damit man später ohne größere Umstände eventuell noch Zeilen zwischenschieben kann.

In welcher Reihenfolge die Zeilen eingegeben werden, ist dem System egal; sie werden automatisch, der Größe der Zeilennummern folgend, *geordnet* und entsprechend bearbeitet, wenn das Programm gestartet wird. Wenn Sie also in Zeile 300 einen Sprung in ein Unterprogramm (GOSUB 20000) eintippen, dann können Sie *sofort* das Unterprogramm schreiben und danach mit Zeile 310 fortfahren.

Wenn Sie eine Programmzeile ändern müssen, können Sie bei moderneren Systemen einfach mit dem Cursor an die betreffende Stelle fahren und die vorhandenen Anweisungen *überschreiben*, ergänzen oder löschen. Andere Modelle haben einen speziellen *Editiermodus* (Sinclair, TI). Die zu ändernde Zeile muß aufgerufen werden, erscheint am unteren Bildschirmrand und kann nur dort abgewandelt werden. Weil dieses Verfahren so umständlich ist, wird es kaum noch verwendet.

Trotzdem müssen Sie beim Überschreiben vorhandener BASIC-Zeilen aufpassen, daß z. B. nichts von den alten Anweisungen ungewollt in die neue Zeile mit hineinrutscht. Eine BASIC-Zeile wird durch Drücken der **RETURN**-Taste eingegeben. Dabei ist es aber völlig gleichgültig, an welcher Stelle der Zeile sich der Cursor befindet. Durch RETURN werden also nicht etwa alle Anweisungen hinter dem Cursor gelöscht.

Bei manchen Systemen (z. B. MSX) kann es beim Überschreiben auch passieren, daß man in die folgende BASIC-Zeile hineinschreibt, wenn das neue Statement länger ist als das alte. Die Zeile,

in die hineingeschrieben wird, nimmt keinen Schaden davon, denn sie ist ja bereits im Speicher abgelegt. Man muß nur wieder darauf achten, daß keine Teile dieser anderen Zeile in die neue Zeile mit hineinkommen.

Ähnliche Vermischungen können entstehen, wenn über schon geschriebene Zeichen hinweg das Programm mit **LIST** auf den Bildschirm geholt wird. LIST sollte deshalb nur im freien unteren Raum des Bildschirms oder nach einem CLEAR HOME gegeben werden.

Beim Ändern der Zeilennummern kommt es auch leicht zu Fehlern. Wenn Sie nur die Zeilennummer überschreiben und dann RETURN geben, haben Sie die Zeile *doppelt,* und zwar unter der neuen und der alten Nummer. Durch das Überschreiben der Zeilennummer wird die überschriebene Zeile nicht im Speicher gelöscht. Eine Zeile läßt sich jedoch sehr leicht aus dem Speicher verbannen, indem man nur die Nummer der zu löschenden Zeile und RETURN eingibt.

Aber das Verdoppeln von Zeilen kann auch zur *Arbeitseinsparung* genutzt werden. In vielen Programmen kommt es vor, daß mehrere ähnliche Zeilen geschrieben werden müssen. Hier können Sie sich die Arbeit erleichtern, indem Sie die Zeile eingeben, nach RETURN mit dem Cursor in die gerade eingegebene Zeile zurückfahren, die Zeilennummer überschreiben und den Inhalt der Zeile abändern. RETURN, und schon ist die nächste Zeile fertig. Dieser Vorgang kann beliebig oft wiederholt werden, und Sie schreiben ein Dutzend Zeilen im Handumdrehen.

Bei fast allen BASIC-Versionen können *mehrere* Befehle unter *einer* Zeilennummer abgelegt werden; sie werden durch *Doppelpunkte* (:) getrennt. Die Länge der BASIC-Zeile ist von System zu System verschieden. Bei Commodore darf sie zwei Bildschirmzeilen (80 Zeichen) lang sein, bei Atari drei Zeilen zu 40 Zeichen und bei MSX-Geräten sogar bis zu 255 Zeichen.

Während das Programm entsteht, stellt sich manchmal heraus, daß mehrere Befehle, die zuerst in einer einzigen Zeile untergebracht wurden, doch verschiedene Zeilennummern benötigen, z. B. um mit einem Sprungbefehl erreichbar zu sein.

Nun ist es aber nicht nötig, alle Befehle noch einmal zu tippen. Verdoppeln Sie einfach die betreffende Zeile durch Überschreiben der Zeilennummer, wie oben beschrieben. Dann *löschen* Sie in jeder der jetzt doppelt vorhandenen Zeilen jeweils einen Teil, und schon haben Sie die Befehle der alten Zeile auf zwei Zeilen verteilt.

Besondere Probleme gibt es, wenn ganze Programmblöcke *verschoben* werden sollen, also mehrere fortlaufende Zeilen andere Nummern erhalten müssen.
Wenn das Programm so kurz ist, daß es auf eine Bildschirmseite paßt, dann gibt es eine ganz simple Lösung. **LIST**en Sie das Programm auf den Bildschirm und überschreiben Sie alle Zeilennummern wie gewünscht. Danach wird mit **NEW** das gesamte Programm im Speicher gelöscht. Aber auf dem Bildschirm ist es noch vorhanden. Sie brauchen also nur mit dem Cursor zur obersten Zeile zu fahren und das Programm, wie es auf dem Bildschirm mit den geänderten Zeilennummern steht, einfach dadurch neu einzugeben, daß Sie *wiederholt* RETURN drücken. Denn nach jedem RETURN springt der Cursor zur jeweils nächsten Zeilennummer. Sie brauchen also nur immer wieder RETURN zu drücken, um das Programm vom Bildschirm an den Speicher zu übergeben.

Meist sind die Programme jedoch länger als die ca. 24 Zeilen, die der Bildschirm faßt. Dann wird das Umstellen von Blöcken schon schwieriger. Selbst wenn BASIC einen **RENUM**-Befehl zum Ändern von Zeilennummern bietet, geht eine solche Umorganisation nicht ohne Probleme ab.

Nehmen wir an, das Programm besteht aus zwei Blöcken mit den Zeilennummern 10 bis 90 und 110 bis 190, die vertauscht werden sollen. Werden die Zeilen 110 bis 190 mit den neuen Zeilennummern 10 bis 90 versehen, dann werden die alten Zeilen 10 bis 90 dadurch überschrieben und gehen verloren.
Der Tausch muß also über eine Zwischenstation abgewickelt werden. Dazu gibt es zwei Möglichkeiten. Die Zeilen 10 bis 90 bekommen irgendwelche Zeilennummern, z. B. 1010 bis 1090. Dann erhalten die Zeilen 110 bis 190 die Nummern 10 bis 90, und danach werden 1010 bis 1090 mit 110 bis 190 überschrieben. Abschließend müssen noch 1010 bis 1090 gelöscht werden, wenn nicht mit dem RENUM-Befehl gearbeitet werden kann, der das automatisch besorgt.

Die andere Methode besteht darin, 110 bis 190 in 15 bis 95 zu verwandeln, dann 10 bis 90 in 110 bis 190 und danach 10, 20 usw. bis 90 zu löschen.

Auf jeden Fall müssen Sie immer, wenn Sie Zeilennummern ändern, das gesamte Programm durchgehen und auch die *Sprungziele* von **GOTO** und **GOSUB** entsprechend *verändern.* Außerdem müssen Sie darauf achten, ob sich nicht vielleicht irgendeine unerwünschte Zeile eingemogelt hat. Und bei allen Tricks – das Austauschen von Programmblöcken bleibt immer ein riskantes Unternehmen, ein üppiger Fehlerquell.

Es ist deshalb viel sinnvoller und arbeitsparender, wenn Sie sich die *Programmstruktur* vorher in etwa überlegen und vielleicht einen *Plan* anfertigen, der die notwendige *Abfolge der Programmblöcke* festlegt, als fröhlich draufloszuprogrammieren und hinterher keine Ordnung mehr in den »Befehlssalat« zu bekommen.

6.2 Fehlersuche

Leider entsteht ein Programm immer in zwei Arbeitsphasen. Im angenehmen Teil werden die Statements, die Arbeitsanweisungen an den Rechner, eingetippt. Und dann, ja dann geht's im zweiten Teil auf Fehlersuche. Jemand kann noch so gut programmieren, es wird doch immer eine Ausnahme sein, wenn ein neues Programm sofort fehlerfrei läuft.

Fehler zu machen, ist also keine Schande. Man sollte nur wissen, wie man sie schnell entlarvt und austilgt.

Je länger und komplexer ein Programm ist, desto größer ist natürlich die Wahrscheinlichkeit, daß sich Fehler einschleichen. Dabei verstecken sich manche so gut, daß sie sich durch alle Tests hindurchmogeln. Selbst in Betriebssystemen von Computern, die heute auf dem Markt angeboten werden, versteckt sich hier und da ein kleiner Fehler.

Beim *Eintippen* des Programms geht es meist schon los. Welcher Computerfreund hat schon eine Sekretärinnenausbildung absolviert? Hinzu kommen die kleinen Stolpersteine der ASCII-Tastatur, mit der uns die Computerhersteller »verwöhnen«.

“Z” und “Y” sind vertauscht; es gibt exotische neue Zeichen, wie die eckige Klammer, das Nummernzeichen (#), hierzulande Raute genannt, das Betragszeichen (@), liebevoll Klammeraffe geheißen, und das Dollarzeichen ($), als Abkürzung für String (Zeichenkette) oft gebraucht. Man muß sich an neue Zeichen für die Operatoren Multiplikation (*) und Division (/) gewöhnen, und auf manche liebgewonnene Schreibmaschinenzeichen muß man verzichten. Da ist es kein Wunder, wenn man sich anfangs häufiger vertippt.

Doch selbst wenn sich die Finger an die ungewohnte Tastatur gewöhnt haben, machen nicht nur dem Programmierneuling Tippfehler oft Probleme. Der Rechner ist ja sehr »kleinkariert«, wenn es um die *Syntax*, die richtige Schreibweise der Befehle, geht.

Wer von seiner alten Klappermaschine Marke Typenhammer gewöhnt war, statt der »1« das kleine L und für die Null das O (Oh!) zu verwenden, der wird sich umstellen müssen. Programmierer erkennt man deshalb in der Regel daran, daß sie jede Null durchstreichen, um sie endgültig gegen Verwechslungen mit dem Oh zu schützen.

Auch auf dem Monitor erscheint die Null mit einem *Schrägstrich*. Doch selbst wenn man für dieses Problem aufmerksam ist, geschieht es immer wieder, daß sich ein Oh als Null einschleicht und umgekehrt, denn auf dem Bildschirm ist der Unterschied bei vielen Systemen kaum zu erkennen. Für ein Programm kann eine solche Verwechslung verhängnisvoll sein, denn die Null ist eine numerische Konstante, während das Oh vom Rechner als Variable behandelt wird.

Viel Verdruß gibt es auch immer wieder mit den *Satzzeichen*, die bei ihrem unauffälligen Aussehen für BASIC doch so sehr wichtig sind. Dabei sehen Komma (,), Semikolon (;) und Doppelpunkt (:) nicht nur bescheiden aus, sie sind auch schlecht zu unterscheiden.

In dem Zusammenhang sind auch die Dezimalzahlen zu erwähnen. In Computerland wird im Gegensatz zu Deutschland ein Dezimalpunkt (.) statt eines Kommas gesetzt. Doch daran sind wir ja gewöhnt, seit wird das Kopfrechnen mit dem Taschenrechner besorgen.

Diese Tippfehler müssen nun nicht unbedingt dazu führen, daß ein Programm überhaupt nicht läuft. Der Rechner kann sie durchaus

akzeptieren, weil sie für ihn ausführbare Kommandos formulieren; nur das gewünschte Ergebnis läßt natürlich auf sich warten.

Wenn ein Programm nicht richtig läuft, ist es also auf jeden Fall ratsam, als erstes auf die oben erwähnten Verwechslungen zu achten.

Bei echten Syntaxfehlern meldet sich der Rechner mit einer *Fehlermeldung*, etwa in der Form »Syntax error at line 110«. Dann ist man natürlich fein raus, weil man sich nur diese BASIC-Zeile auf Schreibfehler anzuschauen braucht. Schreibt man nämlich "GOSUM" statt "GOSUB", dann erkennt der Computer darin kein BASIC-Wort, sondern hält es für eine Variable und erwartet eine Zuweisung (=) oder irgendeine andere Form, in der die Variable vorkommen kann. Und wenn er die nicht vorfindet, dann streikt er eben.

Manche Rechner überprüfen die Syntax schon bei der Eingabe, wenn durch Betätigung der RETURN-Taste die BASIC-Zeile an den Speicher übergeben werden soll. Einfachere Systeme nehmen unbesehen alles an und weisen die Syntax Errors erst aus, wenn das Programm gestartet wird.

Aber wie auch immer, solange das System mit einer Fehlermeldung aufwartet, hat man relativ leichtes Spiel. Selbst die meist schlechten Handbücher zu den Computern enthalten doch immer eine Liste der Fehlermeldungen; und man hat zumindest einen Anhaltspunkt, wonach gesucht werden muß, wenn die Fehlermeldung "NEXT ohne FOR", "Zeile fehlt" oder wie auch immer lautet.

In manchen BASIC-Versionen gibt es auch den Befehl **TRON**. Er bewirkt, daß die gerade bearbeiteten Zeilennummern auf dem Bildschirm angezeigt werden. Tritt ein Fehler auf, hat man die Fehlerstelle dadurch lokalisiert.

Da TRON nicht nur die Zeilennummern anzeigt, sondern das Programm gleichzeitig abarbeitet, also z. B. Ergebnisse auf dem Bildschirm anzeigt, lassen sich damit in gewissem Maße auch logische Fehler aufstöbern. Denn Syntaxfehler auszumerzen – den »Bogen« hat auch ein Anfänger schnell raus. Aber die Stellen zu finden, wo die Programmidee nicht der BASIC-Logik entsprechend umgesetzt wurde, macht schon weit mehr Kopfzerbrechen.

Die meisten Fehler laufen darauf hinaus, daß ein formulierter Befehl bei der Bearbeitung durch den Rechner nicht das bewirkt, was man eigentlich getan haben wollte. Hier ist es sinnvoll, das Programm Zeile für Zeile durchzugehen und im Kopf genau das nachzuvollziehen, was der Rechner machen soll.

Im ersten Durchgang kann man die Programmstruktur überprüfen. Wo sind *Sprünge* (GOTO, GOSUB etc.) vorgesehen, und verzweigen sie zu den gewünschten Zeilen? Hinter einem Sprungbefehl darf in der gleichen Zeile natürlich *kein weiterer Befehl* mehr stehen, denn das Programm kann nie dorthin gelangen, da ja vorher der Sprung ausgeführt wird:

100 GOTO 300:A=20

Die Variable A bekommt nie den Wert 20, weil vorher zur Zeile 300 gesprungen wird; also

100 A=20:GOTO 300

Das gleiche gilt für IF...THEN-Bedingungen. Ist die Bedingung nicht erfüllt, arbeitet BASIC gleich bei der nächsten Zeilennummer weiter:

250 IF A<80 THEN A=0: GOTO 100

Nur wenn die Bedingung *erfüllt* ist, wird auch der Sprung nach Zeile 100 durchgeführt.

Danach sollten die Bedingungen (**IF...THEN...ELSE, ON...GOTO** etc.) *überprüft* werden. Wann ist welche Bedingung erfüllt, und was wird dadurch ausgelöst? Werden die richtigen Variablenzuweisungen getroffen oder die gewünschten Zeilen angesprungen, wenn eine Bedingung gegeben ist? Muß eine Variable, nachdem sie eine Bedingung ausgelöst hat, wieder auf Null zurückgesetzt werden, um die gleiche Bedingung demnächst wieder auszulösen? Kann die bestimmte Bedingung überhaupt erreicht werden? Wirken nur gewollte Faktoren auf das Erreichen der Bedingung ein?

Schließlich kommt die wohl mühsamste Arbeit an die Reihe. Die Variablen müssen überprüft werden. Zuerst sollte man durchsehen, ob jede Variable auch nur an einer Stelle verwendet wird oder für mehrfache Verwendung frei ist.

So kann man für beliebig viele FOR...NEXT-Schleifen immer wieder die *gleiche* Variable als *Schleifenzähler* verwenden; denn wenn die Schleife abgearbeitet ist, wird die Variable und der von ihr gehaltene Wert nicht mehr benötigt. Innerhalb der Schleife darf man aber unter dieser Variablen nicht z. B. das Resultat einer Berechnung aufheben:

```
10 FOR K=10 TO 100 STEP 10
20 K=K*10
30 NEXT K
```

Diese Schleife sollte eigentlich zehnmal durchlaufen werden, wobei K die Werte von 10 über 20, 30 usw. bis 100 annehmen soll. Erst wenn K den Wert 100 erreicht hat, löst das NEXT in Zeile 30 keinen Rücksprung zum FOR in Zeile 10 mehr aus. Doch schon beim ersten Durchgang bekommt K in Zeile 20 den Wert 100, und NEXT gibt den Weg frei. Es muß also eine andere Variable verwendet werden:

```
20 B=K*10
```

Danach kann man verfolgen, welche Werte einzelne Variablen annehmen, und was sie damit im Verlauf des Programms auslösen. Bei Programmstart wird allen numerischen Variablen Null und Stringvariablen ein Leerstring ("") zugewiesen. Bei manchen Systemen haben aber nur die numerischen Variablen den Ausgangswert 0, Feld- und Stringvariablen nicht. In diesen Fällen kann es nötig sein, die Variable bei Programmbeginn erst einmal zu löschen:

```
10 DIM A (20,10)
20 FOR J=0 TO 20:FOR I=0 TO 10:A(J,I)=0:NEXT I:NEXT J
```

Vergißt man dies, können die Variablen obskure Werte halten, die das Programm durcheinanderbringen.

Oft kann es auch sinnvoll sein, die Werte von Variablen auf dem *Bildschirm ausgeben* zu lassen, die sonst nicht zur Ausgabe vorgesehen sind, um zu überprüfen, was innerhalb des Programms abläuft. Man fügt einfach eine Zeile ein, z. B.:

```
32 PRINT D;R(2,2)
```

und der Rechner zeigt bei einem Probelauf, was mit der Variablen D und der Feldvariablen R vor sich geht. Später löscht man diese Zeile einfach wieder.

Wenn dann immer noch nicht das herauskommt, was man vom Programm erwartet, kann man es den Rechner auch abschnittweise bearbeiten lassen. Dazu wird einfach hinter einem bestimmten Programmblock ein *Ende* gesetzt:

163 END

So hat man die Möglichkeit zu sehen, ob bis zu diesem Abschnitt alles richtig läuft.

Mit **GOTO (Zeilennummer)** oder **RUN (Zeilennummer)** kann man bei den meisten Systemen auch die Bearbeitung an der bestimmten Stelle des Programms beginnen lassen. Auf diese Weise wird der Fehlerherd systematisch eingekreist, um dann den wunden Punkt einer strengen logischen Analyse zu unterwerfen.

Der zentrale Gedanke bei der Fehlersuche sollte immer sein:
Der Rechner tut genau das und nur das, was er vorprogrammiert bekommt. Deshalb muß man sich immer wieder fragen: Was bewirkt ein Befehl wirklich? Was will ich, das bewirkt werden soll? Und mit welcher Befehlskombination ist das zu erreichen?

6.3 Optimieren

Die Programmiersprache BASIC ist weit verbreitet, weil sie so leicht zu erlernen ist. BASIC aber glänzt nicht gerade durch hohe Arbeitsgeschwindigkeit. Trotzdem läßt sich aus den Programmen meist noch einiges herausholen, wenn man ein paar einfache Regeln berücksichtigt.

Grundsätzlich kann man sagen, je kürzer ein Programm ist, desto schneller läuft es auch. Mit der Länge ist aber nicht einfach nur die Anzahl der BASIC-Zeilen oder der Zeichen überhaupt gemeint, sondern die Menge von Speicherzellen, die durch das Programm belegt wird.

Viel Platz läßt sich deshalb bei *numerischen Variablen* einsparen. Jeder Integerwert – das sind Ganzzahlen im Bereich von −32768 bis 32767 (Typdeklarationszeichen: %) – belegt nur zwei Byte im Speicher, während Fließkommawerte vier Byte bei einfacher Genauigkeit (!) und gar acht Byte bei doppelter Genauigkeit (#) verbrauchen.

Wenn der verwendete Computer also Typdeklarationen zuläßt und eine Variable in einem bestimmten Programm nur z. B. *ganzzahlige Werte* annehmen kann, dann ist es sinnvoll, die Variable auch entsprechend zu deklarieren. Denn wenn kein Typ bestimmt wird, behandeln die Systeme eine Variable in der Regel als doppeltgenau und verwenden entsprechend viele Speicherzellen für ihre Erfassung.

Kommt eine numerische Konstante häufig in einem Programm vor, dann ist es besser, diesen Wert einer *Variablen* zuzuordnen und fortan die Variable statt des Wertes zu verwenden. Eine Variable verbraucht zwar Platz auf der Variablennamen- und auf der -wertetabelle. Innerhalb des eigentlichen BASIC-Programms belegt sie aber nur *je ein Byte pro Zeichen;* wenn also der Name der Variablen aus einem Zeichen besteht, belegt sie an jeder Stelle, wo sie erscheint, auch nur ein Byte.

Enthält ein Programm nur fünfmal den gleichen doppeltgenauen Fließkommawert, so verbraucht das fünfmal acht, also 40 Byte. Wird dieser Wert einer Variablen zugeordnet, deren Name aus einem einzelnen Zeichen besteht, so werden nur 31 Byte belegt, nämlich zehn Byte für die Zuordnung des numerischen Wertes zu der Variablen innerhalb des BASIC-Programms und acht Byte an den acht Stellen, an denen die Variable statt des Wertes eingesetzt wird. Außerdem belegt die Variable dreizehn Byte in der Namen- und Wertetabelle.

Aus dem gleichen Grund sollten lange BASIC-Funktionen durch *Zuordnung zu einer Variablen* vermieden werden. Statt bei der Abfrage des Steuerknüppels mehrmals STICK(1) zu schreiben, wird V=STICK(1) zugeordnet und dann nur noch die Variable verwendet. Statt z. B. mehrfach PEEK (55323) zu programmieren, wird einmal J=PEEK (55323) zugeordnet und danach nur noch J verwendet. Statt mehrmals im Programm SAVE "D:ZWISCHENKOPIE" zu schreiben, wird einmal N$="ZWISCHENKOPIE" zugeordnet und in der Folge nur noch SAVE N$ geschrieben.

Variablennamen sollten möglichst *kurz* gehalten werden. Beim Programmieren kann es hilfreich sein, längere Variablen zu verwenden, deren Name an ihre Funktion erinnert. Jedoch belegt jedes Zeichen der Variablen einen Speicherplatz; und wenn eine

solche Variable mehrmals in einem Programm auftaucht, dann kann sich das ganz schön addieren.

Da die meisten Heimcomputer ohnehin nur die beiden ersten Zeichen eines Variablennamens unterscheiden und den Rest nur für den Programmierer mitschleppen, sollten Sie alle Variablennamen entsprechend kürzen, sobald Ihr Programm fehlerfrei läuft.

Anschließend sollten Sie auch die *Variablennamentabelle säubern.* Diese Tabelle liegt in einem Speicherbereich außerhalb des eigentlichen BASIC-Programms und enthält alle Variablennamen und einen Hinweis (Vektor), wo im Speicher der zu dieser Variablen gehörende Wert abgelegt ist; das ist die *Variablenwertetabelle* bei numerischen Variablen oder die *Stringtabelle* bei Stringvariablen.

In die Variablennamentabelle werden alle Variablen fortlaufend aufgenommen, also auch Variablen, die einmal gebraucht, später aber wieder aus dem Programm herausgenommen worden sind. Auch jeder Tippfehler, der zufällig einem zulässigen Variablennamen entspricht, landet in dieser Tabelle. Da kommt schnell eine Menge Abfall zusammen. Und wie bekommt man hier wieder Ordnung hinein?

Bessere Heimcomputer erlauben die Speicherung von Programmen sowohl im ASCII- wie auch im *tokenisierten* Format. Beim letzteren wird das Programm in eine Art Kurzschrift übertragen. Die Variablennamentabelle wird dabei so gespeichert, wie sie gerade ist.

Wird ein Programm im ASCII-Format aufgezeichnet, so wird das Listing, Zeichen für Zeichen, in ASCII-Codes umgesetzt, auf das Speichermedium geschrieben; die Tabellen werden nicht mitgespeichert.

Um die Variablennamentabelle zu bereinigen, muß das Programm im ASCII-Format gespeichert werden; **NEW** löscht den Speicher des Rechners mit allen Tabellen; danach wird das Programm wieder geladen. Die Variablennamentabelle wird dabei neu geschrieben und enthält dann nur die tatsächlich im Programm auftretenden Variablennamen.

Auch **REM**-Zeilen sind bei vielen Hobbyprogrammierern beliebt, weil sie den Überblick erleichtern. Für den Rechner stellen sie schieren Ballast dar. *Löschen* Sie deshalb alle REM-Bemerkungen, sobald Ihr Programm fehlerfrei läuft.

Anfänger gehen meist auch recht üppig mit *Dimensionierungen* um, so nach dem Motto: »Darf's ein wenig mehr sein?« Mit der Dimensionierung wird entsprechend viel Platz im Speicher reserviert. Wenn Sie dimensionierte Variablen verwenden, dann dimensionieren Sie *exakt nach Bedarf.* Denken Sie auch daran, daß die Indizes mit 0 beginnend gezählt werden; also belegt DIM A (9,9) Platz für zehn mal zehn (= 100!) Elemente der Variablen A.

Werden die beiden Indizes nur um 1 erhöht, würde A auf (10,10) dimensioniert. Dadurch müßte Platz für 21 weitere Elemente reserviert werden. Da A nicht weiter deklariert ist, wird sie als doppeltgenaue Fließkommavariable behandelt, die acht Byte belegt. 21 zusätzliche Elemente blockieren damit 168 Speicherplätze.

Ob Sie hohe oder niedrige Zeilennummern verwenden, das macht für den Rechner keinen Unterschied, denn jede Zeilennummer belegt immer zwei Byte. Aber jedes Programm, das Sprunganweisungen (GOTO, GOSUB) enthält, ist wirksamer, wenn es aus möglichst wenigen BASIC-Zeilen besteht.

Auf dem Bildschirm mag es etwas verwirrender aussehen, aber Ihr Programm läuft schneller, wenn Sie möglichst *viele Statements* hinter einer Zeilennummer unterbringen. Statt

```
10 FOR J=0 TO 99
20 FOR I=0 TO 99
30 POKE I*8+J,0
40 NEXT I
50 NEXT J
```

schreiben Sie besser

```
10 FOR J=0 TO 99:FOR I=0 TO 99:POKE I*8+J,0:NEXT
I:NEXT J
```

Das Programm läuft deshalb *schneller,* weil der Rechner bei jeder Sprunganweisung nach der bestimmten Zeilennummer suchen muß. Das tut er, indem er das gesamte Programm von vorn durchgeht und jede einzelne Zeilennummer, die er findet, mit der für den Sprung bestimmten Zeilennummer vergleicht. Da ist klar: Je *weniger* Zeilennummern ein Programm hat, desto schneller kommt der Rechner damit durch.

Aus dem gleichen Grund sollten auch Zeilen, die mehrfach angesprungen werden, möglichst *niedrige* Zeilennummern bekommen.

Viele Programme sind so aufgebaut, daß sie mit einem aufwendigen Vorspann beginnen, der vielleicht die Zeilen 10 bis 490 belegt und nur bei Programmbeginn einmal durchlaufen wird. Das eigentlich arbeitende Programm, das womöglich mehrmals durchlaufen werden muß, beginnt dann bei Zeile 500. Bei jedem Sprung an diese Stelle muß der Rechner somit alle Zeilennummern von 10 bis 490 durchsehen, bis er die 500 findet. Wollen Sie ihm das wirklich zumuten?

Die beiden Beispielprogramme sind für MSX geschrieben. Der Effekt tritt aber bei anderen Rechnern entsprechend auf. In der Version LANGSAM.MSX benötigt der Rechner 105,8 Sekunden, um das Programm abzuarbeiten:

```
0  REM LANGSAM.MSX * 105,8 Sekunden *
10 CLS:DIM VARIABLE(30,30)
20 REM ***********************
30 REM *                     *
40 REM *   Listing für MSX   *
50 REM * von Karl-Heinz Koch *
60 REM *                     *
70 REM ***********************
80 REM
100 REM * Vorspann *
110 PRINT"A",3.010203040506#
120 PRINT"B",3.010203040506#
130 PRINT"C",3.010203040506#
140 PRINT"D",3.010203040506#
150 PRINT"E",3.010203040506#
160 PRINT"F",3.010203040506#
170 PRINT"G",3.010203040506#
180 PRINT"H",3.010203040506#
190 PRINT"I",3.010203040506#
300 REM * FOR-NEXT-Scheleife *
310 FOR SPALTE=0 TO 9
320 FOR ZEILE=0 TO 9
330 VARIABLE(SPALTE,ZEILE)=SPALTE*ZEILE
340 NEXT ZEILE
350 NEXT SPALTE
400 REM * Block zwei *
410 ZAEHLER=ZAEHLER+1
420 IF ZAEHLER=100 THEN GOTO 500
430 GOTO 300
500 REM * Auswertung *
510 BEEP
520 PRINT"FERTIG"
530 END
```

In diesem Programm ist so ziemlich alles falsch gemacht, was man nur falsch machen kann. Es macht sich mit 695 Byte im Speicher

breit. Natürlich ist das Beispiel übertrieben, aber es dient ja auch nur der Demonstration.

In der Version SCHNELL.MSX ist das Programm konsequent »abgespeckt« worden; es bescheidet sich jetzt mit 268 Byte, also 427 Byte weniger. Der Rechner bedankt sich, indem er 15,2 Sekunden schneller damit fertig wird:

```
0  REM SCHNELL.MSX * 90,6 Sekunden *
10 CLS:DIM V%(9,9):W=3.010203040506#:GOSUB 11
00
330 FOR S=0 TO 9:FOR Z=0 TO 9:V%(S,Z)=S*Z:NEX
T Z:NEXT S:K=K+1:IF K=100 THEN 510
430 GOTO 330
510 BEEP:PRINT"FERTIG":END
1100 PRINT"A",W
1200 PRINT"B",W
1300 PRINT"C",W
1400 PRINT"D",W
1500 PRINT"E",W
1600 PRINT"F",W
1700 PRINT"G",W
1800 PRINT"H",W
1900 PRINT"I",W
2000 RETURN
```

7 Tabellen

Code	Zeichen	Code	Zeichen	Code	Zeichen	Code	Zeichen
0	NUL	32	Leerzeichen	64	@	96	\
1	SOH	33	!	65	A	97	a
2	STX	34	''	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	—	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	‘	124	'
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	—	127	DEL

Tabelle 1
Der Sieben-Bit-ASCII-Code in der internationalen Version. Heimcomputer richten sich weitgehend nach diesem Standard. Die Codes 0 bis 31 bedeuten Drucker-Steuerungszeichen

Tabelle 2
Der Bildschirmcode für den Commodore. Um z. B. ein »K« in die linke obere Ecke des Bildschirms zu schreiben, muß POKE 1024,11 eingegeben werden. Die Codes 128–255 ergeben die gleichen Zeichen in inverser (negativer) Darstellung: POKE 1024,11 + 128

Zeichen	Code	Zeichen	Code	Zeichen	Code
@	0	V	22	,	44
A	1	W	23	–	45
B	2	X	24	.	46
C	3	Y	25	/	47
D	4	Z	26	0	48
E	5	[	27	1	49
F	6	£	28	2	50
G	7	]	29	3	51
H	8	↑	30	4	52
I	9	←	31	5	53
J	10	SPACE	32	6	54
K	11	!	33	7	55
L	12	"	34	8	56
M	13	#	35	9	57
N	14	$	36	:	58
O	15	%	37	;	59
P	16	&	38	<	60
Q	17	'	39	=	61
R	18	(	40	>	62
S	19	)	41	?	63
T	29	*	42		
U	21	+	43		
	64		86		108
	65		87		109
	66		88		110
	67		89		111
	68		90		112
	69		91		113
	70		92		114
	71		93		115
	72		94		116
	73		95		117
	74	SPACE	96		118
	75		97		119
	76		98		120
	77		99		121
	78		100		122
	79		101		123
	80		102		124
	81		103		125
	82		104		126
	83		105		127
	84		106		
	85		107		

0	SCHWARZ	8	ORANGE
1	WEISS	9	BRAUN
2	ROT	10	HELLROT
3	TÜRKIS	11	GRAU 1
4	VIOLETT	12	GRAU 2
5	GRÜN	13	HELLGRÜN
6	BLAU	14	HELLBLAU
7	GELB	15	GRAU 3

Tabelle 3
Farbcodes für Commodore. Um z. B. die Farbe des Zeichens in der linken oberen Ecke des Bildschirms nach Violett zu verändern, muß POKE 55296,4 eingegeben werden

Code	Farbe	Code	Farbe	Code	Farbe	Code	Farbe
0	Transparent	4	Dunkelblau	8	Rot	12	Dunkelgrün
1	Schwarz	5	Hellblau	9	Hellrot	13	Magenta
2	Grün	6	Dunkelrot	10	Dunkelgelb	14	Grau
3	Hellgrün	7	Cyan	11	Hellgelb	15	Weiß

Tabelle 4
Farbwerte für MSX

Tabelle 5 *(auf Seite 155)*
Farbwerte für Atari. Um z. B. die Farbe für COLOR 1 in Flieder zu verwandeln, muß POKE 708,66 eingegeben werden



Tabelle 6 *(auf den Seiten 156/157)*

Verschiedene BASIC-Dialekte im Vergleich der wichtigsten Befehle

Zeichenerklärung:

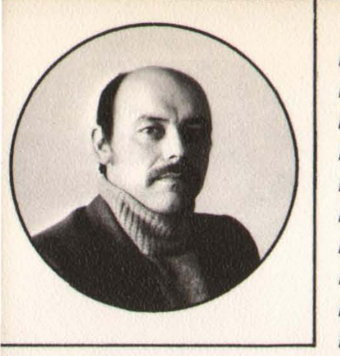
A Ausdruck – z. B. 4+3 oder 2*(DA-6.5)
K beliebige Konstante, numerisch oder String-
N numerische Variable oder Feldvariable
V beliebige Variable, numerisch, String- oder Feld-
Z eine Zeilennummer
N$ Stringvariable
n ein Index
x eine Spaltennummer auf dem Bildschirm
y eine Zeilennummer auf dem Bildschirm

Schwarz	Mittel-		relativ gleichmäßige Grau-Skala bis fast				Weiß
0	2	4	6	8	10	12	14
Purpur	Rehbraun	Curry	Schmutzig-	leicht grünliche Gelbtöne			Eierschale
16	18	20	22	24	26	28	30
Rot	Indischrot	gedecktes	Orange	Orange	über Gelb bis weißlich		
32	34	36	38	40	42	44	46
Glutrot	feurig	Altrosa	Lachs		Abstufungen von Rosa		
48	50	52	54	56	58	60	62
Violett	Flieder			von Hellviolett über Lila nach weißlich Lila			
64	66	68	70	72	74	76	78
Blau			von Blauviolett über Flieder nach Blaßlila				
80	82	84	86	88	90	92	94
Cyan			gleichmäßige Skala bis fast Weiß				
96	98	100	102	104	106	108	110
Cyan	leicht graustichig			fast identisch mit der vorstehenden Reihe			
112	114	116	118	120	122	124	126
Preußisch				Skala etwas dunstiger Blautöne			
128	130	132	134	136	138	140	142
Dunkelblau		Mittelblau		von Taubenblau über Stahlblau nach Eisblau			
144	146	148	150	152	154	156	158
Mittelgrün			über Hellgrün und Giftgrün ins leicht türkisierende Weiß				
160	162	164	166	168	170	172	174
Mittelgrün	Maigrün		der vorstehenden Reihe ähnlich, jedoch etwas greller				
176	178	180	182	184	186	188	190
Dunkelgrün			gleichmäßige Skala mit höherem Gelbanteil				
192	194	196	198	200	202	204	206
Moosgrün			wie vorstehend, jedoch mit noch größeren Gelbanteilen				
208	210	212	214	216	218	220	222
Braun	Oliv	Grünbeige		gelblich schmutzige Grüntöne			
224	226	228	230	232	234	236	238
Karminrot			von Rehbraun über Curry und schmutzige Gelbtöne				
240	242	244	246	248	250	252	254

		Atari	Apple	Commodore C20/64	IBM-PC	MSX	Sinclair Spectrum	PC 1500 Sharp	TI 99/4A
Zuweisung		LET V=A	LET V=A	LET V=A	LET V=A	LET V=A	LET V=A	LET V=A	LET V=A
Dimensionierung		DIM V	DIM V	DIM V	DIM V	DIM V	DIM V	DIM V	DIM V
Daten		DATA K	DATA K	DATA K	DATA K	DATA K	DATA K	DATA K	DATA K
Daten lesen		READ V	READ V	READ V	READ V	READ V	READ V	READ V	READ V
Strings	Länge	LEN N$	LEN N$	LEN N$	LEN N$	LEN N$		LEN N$	LEN N$
	linker Teil	N$ (n,n)=A$	LEFT$ (N$,n)	LEFT$ (N$,n)	LEFT$ (N$,n)	LEFT$ (N$,n)		LEFT$ (N$,n)	SEG$ (N$,n,n)
	rechter Teil	N$ (n,n,)=A$	RIGHT$ (N$,n)	RIGHT$ (N$,n)	RIGHT$ (N$,n)	RIGHT$ (N$,n)		RIGHT$ (N$,n)	SEG$ (N$,n,n)
	mittlerer Teil	N$ (n,n)=A$	MID$ (N$,n,n)	MID$ (N$,n,n)	MID$ (N$,n,n)	MID$ (N$,n,n)		MID$ (N$, n, n)	SEG$ (N$, n, n)
Sprung		GOTO Z	GOTO Z	GOTO Z	GOTO Z	GOTO Z	GOTO Z	GOTO Z	GOTO Z
bedingter Sprung		GOTO A	ON A GOTO	ON A GOTO	ON A GOTO	ON A GOTO		ON A GOTO	ON A GOTO
Unterprogramm	Sprung	GOSUB Z	GOSUB Z	GOSUB Z	GOSUB Z	GOSUB Z	GOSUB Z	GOSUB Z	GOSUB Z
	bedingter Sprung	GOSUB A	ON A GOSUB	ON A GOSUB	ON A GOSUB	ON A GOSUB		ON A GOSUB	ON A GOSUB
	Rücksprung	RETURN	RETURN	RETURN	RETURN	RETURN	RETURN	RETURN	RETURN
Bedingung	Verzweigung	IF A THEN Z	IF A THEN Z	IF A GOTO Z	IF A THEN Z	IF A THEN Z	IF A THEN Z	IF A GOTO Z	IF A THEN Z
	Anweisung	IF A THEN A	IF A THEN A	IF A THEN A	IF A THEN A	IF A THEN A	IF A THEN A	IF A THEN A	IF A THEN A
	strukturiert				IF A THEN A ELSE A	IF A THEN A ELSE A			IF A THEN A ELSE A

Schleife		FOR N=A TO A NEXT N	FOR N=A TO A NEXT N	FOR N=A TO A NEXT N	FOR N=A TO A NEXT N	FOR N=A TO A NEXT N	FOR N=A TO A NEXT N	FOR N=A TO A NEXT N	FOR N=A TO A NEXT N
Schrittweite		STEP A	STEP A	STEP A	STEP A	STEP A	STEP A	STEP A	STEP A
Bildschirm	Ausgabe	PRINT A	PRINT A	PRINT A	PRINT A	PRINT A	PRINT A	PRINT A	PRINT A
	Position	POSITION X,Y,	HTAB X VTAB Y		LOCATE X,Y	LOCATE X,Y	PRINT (Y,X)		DISPLAY AT (X,Y)
ASCII – Zeichen		CHR$ (A)	CHR$ (A)	CHR$ (A)	CHR$ (A)	CHR$ (A)	CHR$ (A)	CHR$ (A)	CHR$ (A)
Zeichen – ASCII		ASC (N$)	ASC (N$)	ASC (N$)	ASC (N$)	ASC (N$)	CODE (N$)	ASC (N$)	ASC (N$)
Ganzzahl		INT (A)	INT (A)	INT (A)	INT (A)	INT (A)	INT (A)	INT (A)	INT (A)
Zufall	Ausgangswert			RND (-A)	RANDOMIZE (A)	RND (-A)	RANDOMIZE (A)	RANDOM	RANDOMIZE
	Zufallszahl	RND (0)	RND (1)	RND (A)	RND (A)	RND (A)	RND (A)	RND (A)	RND
Programm	Starten	RUN	RUN	RUN	RUN	RUN	RUN	RUN	RUN
	Löschen	NEW	NEW	NEW	NEW	NEW	NEW	NEW	NEW
	auf dem Bildschirm zeigen	LIST	LIST	LIST	LIST	LIST	LIST	LIST	LIST
	auf dem Drucker ausgeben	LIST "P"	LLIST		LLIST	LLIST			LIST "P"

Register



Karl-Heinz Koch sammelte Erfahrungen als Pädagoge und arbeitet heute als freier Publizist. Aus reiner Neugierde hat er sich selbst das Programmieren beigebracht und ist inzwischen mit zahlreichen Artikeln in Computerzeitschriften und mehreren Büchern hervorgetreten, in denen er immer wieder zeigt: Programmieren kann jeder lernen.

ht-DIE REIHE, DIE ZUR SACHE KOMMT.

- BASIC-Grundlagen für jeden.
- Grafik, Farbe und Bewegung: Wie aus Zahlen Bilder werden.
- Spielfeld, Spieler, „Feinde": Wie Sie dem Computer die Spielregeln beibringen.
- Spieldauer, Wertung und Vertonung: Was ein Spiel lebendig macht.
- Zahllose Anregungen und viele komplette Programme für Atari, Commodore, MSX.

Lernen Sie die faszinierenden Möglichkeiten des Heimcomputers spielend kennen. Dieser leicht verständliche Band zeigt Ihnen einen schöpferischen Weg zu unbegrenztem Freizeitspaß.

ISBN 3-581-66526-3 DM 8.80



praktische ratgeber
humboldt-taschenbücher